Diss. ETH No. 25568

Idealizations of Practical Cryptographic Building Blocks

A thesis submitted to attain the degree of

Doctor of Sciences of ETH Zurich
(Dr. sc. ETH Zurich)

presented by

Christian Badertscher
MSc ETH in Computer Science, ETH Zurich

born on August 24, 1988
citizen of Zäziwil BE, Switzerland

accepted on the recommendation of

Prof. Dr. Ueli Maurer, examiner
Prof. Dr. Aggelos Kiayias, co-examiner
Prof. Dr. Srdjan Čapkun, co-examiner

2018

Acknowledgements

First and foremost, I would like to thank my advisor Ueli Maurer for the opportunity to do a PhD in the exciting field of cryptography and in a environment that is driven by the quest of finding convincing answers to fundamental problems in cryptography. Our research discussions deeply inspired me and his theoretical and abstract way of thinking led to many interesting insights and shaped my ability to solve problems.

Sincere thanks go to Aggelos Kiayias and Srdjan Čapkun for co-refereeing this thesis.

I would like to take the opportunity to thank all my co-authors during my PhD studies. For the exciting former and hopefully future collaborations and the fruitful exchange of ideas, I thank Fabio Banfi, Sandro Coretti, Juan Garay, Peter Gaži, Daniel Jost, Aggelos Kiayias, Chen-Da Liu Zhang, Christian Matt, Ueli Maurer, Phillip Rogaway, Alexander Russell, Björn Tackmann, Daniel Tschudi, and Vassilis Zikas.

Special thanks go to Vassilis Zikas for the many great formal and informal discussions on various aspects of life, MPC, and of course blockchain protocols. Many thanks go to Martin Hirt for being a valuable discussion partner and an amazing lecturer throughout all these years.

I thank Joel Alwen, Ämin Baumeler, Grégory Demay, Robert Enderlein, Gian-Pietro Farina, Thomas Holenstein, Christoph Lucas, Marta Mularczyk, Christopher Portmann, Pavel Raykov, Guilherme Rito, Gregor Seiler, Stefan Wolf, and Jiamin Zhu for interesting discussions, passionate table-soccer games, and exciting Magliaso retreats, and Beate Bernhard, Claudia Günthart, and Denise Spicher for the administrative support.

Needless to say, this thesis would not have been possible without the love of Anna, the invaluable encouragement of my family, and the support of my dear friends. Merci!

Abstract

Security definitions are at the core of cryptographic research. Their importance stems from the fact that they enable security proofs of protocols in a mathematically rigorous way. More specifically, one has to develop formal models and security notions such that the derived security guarantees of a protocol are sound and convincing. Most existing security definitions are property-based or game-based, which means that a protocol is secure if it fulfills a postulated set of requirements. While this approach seems fine at first sight, it has several drawbacks. The reason is that such properties are formulated with respect to an attacker with a defined set of capabilities. However, if a protocol is used in an application where an attacker has more influence than previously assumed, all proven guarantees turn out to be void.

A much better approach is to define security as a construction and in particular to specify what ideal system or module a cryptographic protocol achieves in any possible context. Turned around, any application can be assured that running the protocol is as if the ideal system was available in the first place. For example, a protocol for secure communication could be expected to emulate an ideal secure channel, i.e., a system that allows to transmit a message from a sender to a receiver and no one except for the intended recipient should learn the contents of the message. A security proof consists of showing that a protocol, based on certain assumptions, is indeed indistinguishable from the ideal system. Such proofs are typically quite involved and more complex than property-based proofs.

In this thesis, we follow the above methodology to study practically relevant cryptographic problems for which it is important to have clean security statements that hold in any possible context. This thesis contributes to the following areas:

In the realm of secure communication, we formulate which ideal systems are achieved by protocols that are based on symmetric primitives. We further show the limits of such protocols by showing that no protocol can perfectly emulate a secure channel only based on a shared secret key and insecure communication. We further extend our study to communication protocols that are based on public-key primitives.

In the realm of secure outsourced storage, we develop a novel framework to reason about the achieved security of data-outsourcing schemes. We show which ideal systems are desired and possible to achieve. We hereby observe that existing definitions are often weaker than what would be needed by applications and we show how to improve existing protocols to achieve a higher level of security.

In the context of digital signature schemes, we present a novel way to capture their security following the above methodology and prove that this novel view is equivalent to the standard game-based definition.

Finally, in the context of blockchain protocols, we show which ideal functionality is achieved by the Bitcoin blockchain. Since numerous emerging applications rely on the security provisions of blockchain protocols, identifying the ideal module which any such application can rely on is of great practical importance.

Zusammenfassung

Sicherheitsdefinitionen sind ein zentraler Bestandteil in der kryptographischen Forschung. Die grosse Bedeutung stammt vom Wunsch, mathematisch beweisbare Sicherheit von Protokollen zu zeigen. Um dies zu erreichen, müssen Modelle und Definitionen entwickelt werden, so dass ein mathematischer Beweis möglich und dessen Schlussfolgerung überzeugend ist. Viele existierende Sicherheitsdefinitionen sind im Kern eine Liste von wünschenswerten Eigenschaften, die ein Protokoll zu erfüllen hat. Ein solches Vorgehen scheint auf den ersten Blick sinnvoll, hat jedoch einige negative Apsekte. Protokolleigenschaften sind meistens relativ zu einem Angreifer mit fest definierten Möglichkeiten formalisiert. Wenn nun ein Protokoll in einer speizifischen Applikation genutzt wird, in welcher der Angreifer mehr Möglichkeiten hat Einfluss zu nehmen, so ist die Liste der Eigenschaften nutz- und die Sicherheitsbeweise wertlos.

Ein viel besserer und modernerer Ansatz ist stattdessen, dass man formalisiert, was für eine ideale Funktionalität ein Protokoll emuliert in einem beliebigen Kontext. Man gibt also genau an, was beliebige Applikationen von einem Protokoll erwarten können, indem man die Schnittstellen abstrahiert und das Protokoll als idealisiertes Modul modelliert, welches ein bestimmtes Verhalten garantiert. Zum Beispiel könnte man von einem sicheren Netzwerkprotokoll erwarten, dass es sich verhält wie ein ideales Modul zum Senden von Nachrichten, wo niemand ausser dem Sender und Empfänger die Nachricht sehen kann. Ein Sicherheitsbeweis besteht nun daraus, zu zeigen, dass ein Protokoll das ideale Modul implementiert. Solche Beweise sind typischerweise viel komplexer, als nur gewisse Eigenschaften zu zeigen.

In dieser Dissertation wenden wir diese Methodologie auf praktisch relevante Problemstellungen an, bei denen es besonders wichtig ist, dass

man Sicherheitsgarantien geben kann, die in beliebigen Kontexten gelten. Dazu gehören die folgenden Themenbereiche:

Im Bereich der sicheren Kommunikation zeigen wir, welche idealen Module von symmetrischen Protokollen wie TLS implementiert werden sollen und zeigen auch, dass es nicht möglich ist, ein in jeder Hinsicht perfekt sicheres Kommunikationsprotokoll zu entwickeln. Wir wenden die gleiche Methodik auf asymmetrische Protokolle an (basierend auf Public-Key Kryptographie).

Im Bereich der sicheren externen Datenspeicherung entwickeln wir ein neues Rahmenwerk um die Sicherheit von Protokollen zu erfassen. Wir zeigen, welche idealen Module möglich und wünschenswert sind. Dabei entdecken wir, dass bisherige, auf Eigenschaften basierte Definitionen, oftmals zu schwach sind. Wir zeigen, wie man existierende Protokolle verstärken kann, um mehr Sicherheit zu erreichen.

Im Bereich der digitalen Signaturen präsentieren wir eine alternative Möglichkeit, die Sicherheit zu fassen und zeigen, dass unsere Darstellung äquivalent zu existierenden Definitionen ist.

Schliesslich zeigen wir, welche ideale Funktionalität von Blockchains, insbesondere der Bitcoin Blockchain, implementiert wird. Dies ist äusserst wichtig, da unzählige und verschiedenste Applikationen auf der Sicherheit der Blockchain basieren. Dieses ideale Modul zu identifizieren und zu beweisen, dass Bitcoin dieses tatsächlich implementiert, ist von enormer praktischer Relevanz.

Contents

Chapter 1

Introduction

"Imagination should be used, not to escape reality, but to create it."

- Colin Wilson, 1962.

1.1 Motivation

A fundamental task in cryptographic research is the design of secure protocols and the formal study of protocol properties. This includes precise statements on what guarantees cryptographic protocols achieve when used in a larger application. For example, if an application requires a shared secret key between Alice and Bob, a natural approach is that they initiate a key agreement sub-protocol. The resulting key is then provided to the application and used, for example, to protect future communication. What we intuitively require from the key exchange sub-protocol is that the resulting output is as good as if a uniformly random shared secret key had been available to the parties in the first place.

In cryptography, the security of protocols is usually captured by technical definitions that are prevalently motivated by an application story that justifies the utility and the line of reasoning that led to the definition. This justification often leads to the conclusion that a protocol that provably satisfies a security definition, can be safely used in intended applications. However, this final conclusion is typically not argued formally

and it is not sound in general: Many existing security definitions for cryptographic protocols fall short in guaranteeing the security of intended applications, and in extreme cases are even flawed and lead to successful attacks. We hence face a situation in cryptography where a mismatch between formal definitions and effective usage of a protocol cannot be excluded.

To overcome this situation requires a fundamentally different methodology. A protocol has to be seen as a construction that transforms an assumed setting into an idealized setting. In the example of key agreement, the assumed setting could be an authenticated communication channel and the goal of a key agreement protocol is to construct an ideal, uniformly random shared secret key out of the communication channel. A core property of a constructive view is composability: in any application that assumes a shared secret key, for example symmetric encryption, it is safe to replace the ideal key by the construction achieving it.

Consequently, a foundational task in cryptographic research is to build a library of important constructions that formalizes the security of fundamental cryptographic building blocks and that establishes a solid basis to create larger applications in a modular way based on the achieved idealizations. The main objective of this thesis is to contribute to such a library in the area of secure communication, outsourced storage, and blockchain protocols.

1.2 Security Notions and Applications

As briefly mentioned above, classical cryptographic definitions, including the ones we study in this thesis, do not capture in which contexts a scheme satisfying them can securely be used, leaving an undesirable gap between a scheme's security properties and its use. They consider a specific attack model and give certain capabilities to an adversary that tries to win some game, but it is not *a priori* clear which capabilities an adversary has in a particular application, or even what the final goal would be. To illustrate our basic point, consider the standard notions for encryption schemes, IND-CPA and IND-CCA. While IND-CCA is stronger, it is not obvious in which applications an IND-CCA encryption scheme is needed and where IND-CPA would suffice. These considerations are highly security-relevant. For complex protocols like TLS or IPSec, one has to make sure that any

overall attack can be translated to an attack against the CPA or CCA game or another hardness assumption; only then the protocol is sound. But such analyses are complex and cannot be reused for the analysis of other protocols or attack models. This directly calls for a more elegant and modular approach.

Our approach. The basis of our approach is the constructive cryptography framework (CC) developed by Maurer and Renner [Mau12, MR11]. A central concept in constructive cryptography is to model cryptographic settings by means of *resources*. Resources model both the real setting in which a protocol is executed (e.g., a protocol makes use of a secret key and a channel) and the ideal abstraction that the parties want to achieve (e.g., a secure channel). The role of a cryptographic protocol is to construct the desired resource from the assumed ones. We will formally define it later. An important property of this construction notion is its composability. A constructed resource can be used in any other construction as an assumed resource in any further application and security is preserved if this assumed resource is replaced by the construction achieving it. We note that such a rigorous approach stands in sharp contrast to game-based security definitions.

Using the framework of constructive cryptography, we want to bridge the gap between technical security definitions of cryptographic primitives and the security of intended applications having these primitives as a building block. We want to find the appropriate idealizations and show which resources can be constructed in a given setting, with a special focus on practically relevant building blocks in order to shed new light on existing definitions and to spot and rectify potential weaknesses and provide better constructions if needed.

1.3 Overview and Contributions

This thesis makes contributions in various fields of cryptography that are currently relevant in practice. This includes the study of communication primitives, outsourced storage protocols, and blockchain protocols. In the following, we highlight the main contributions and give pointers to the relevant sections and the papers they are based on.

1.3.1 Secure Communication Primitives

In this thesis, Chapters 3 to 5 are devoted to investigate cryptographic primitives that are used to secure the communication over the internet. Two cases are discussed: symmetric primitives and asymmetric primitives. The results in Chapters 3 to 5 are based on the publications [BMM+15a, BMM+15b] and [BBM18], respectively.

Authenticated Encryption. Authenticated encryption (AE) schemes were introduced to provide both authenticity and confidentiality of sent messages with the motivation to offer a simpler interface to applications and better performance compared to generic compositions of encryption and authentication mechanisms. Due to their wide adoption and importance in practical protocols like TLS 1.3, it seems important to idealize the goal of authenticated encryption schemes.

We therefore present a novel basic channel abstraction, an *augmented secure channel* (ASC), that allows a sender to send a receiver messages consisting of two parts, where one is privacy-protected and both are authenticity-protected. We formalize this idea in constructive cryptography and provide a construction of this kind of channel generically based on authenticated-encryption schemes. As an immediate application, we can look at the development of TLS 1.3 and observe that the criterion by which all the drafts can be judged actually becomes quite simple: do they construct an ASC? According to this measure, we can investigate different drafts of TLS 1.3 and see what is achieved through the lens of provable security.

Robust AE and the limits of symmetric cryptography. Robust authenticated encryption (RAE) can be seen as an extension to AE and is a primitive for symmetric encryption that allows to flexibly specify the ciphertext expansion, i.e., how much longer the ciphertext is compared to the plaintext. For every ciphertext expansion, RAE aims at providing the best-possible authenticity and confidentiality. To investigate whether this is actually achieved, we characterize exactly the guarantees symmetric cryptography can provide for any given ciphertext expansion. We formulate this idealization again in the constructive cryptography framework.

Our characterization reveals not only that RAE reaches the claimed goal, but also, contrary to prior belief, that one cannot achieve full confidentiality without ciphertext expansion. This provides new insights into the limits of symmetric cryptography. Moreover, we provide a rigorous treatment of two previously only informally stated additional features of RAE; namely, we show how redundancy in the message space can be exploited to improve the security and we analyze the exact security loss if multiple messages are encrypted with the same nonce.

Signcryption. Our study on communication primitives is concluded by considering the asymmetric (or public-key) counterpart of authenticated encryption: Signcryption. This is a public-key cryptographic primitive, originally introduced by Zheng in 1997, that allows parties to establish secure communication without the need of prior key agreement. Instead, a party registers its public key at a certificate authority (CA), and only needs to retrieve the public key of the intended partner from the CA before being able to protect the communication. Like authenticated encyrption, signcryption schemes provide both authenticity and confidentiality of sent messages. Although introduced two decades ago, the question which security notions of signcryption are adequate in which applications has still not reached a fully satisfactory answer.

We again conduct a constructive analysis to derive what signcryption schemes should ideally achieve. We identify the goal of signcryption as a gracefully-degrading secure network, which is basically a network of independent parties that allows secure communication between any two parties. However, when a party is compromised, its respective security guarantees are lost, while all guarantees for the remaining users remain unaffected. A signcryption scheme should enable a construction of this network solely based on a CA and an insecure network. As part of our study, we derive the game-based security notions that are sufficient for the construction to be achieved, which helps to understand what the "standard" game-based notions of signcryption should be.

1.3.2 Secure Outsourced Storage

In Chapter 6 of this thesis, we discuss the problem of securely outsourcing storage to an external service provider. Indeed, the security of data

outsourcing mechanisms has become a crucial aspect of today's IT infrastructures and they are the cryptographic foundations of real-world applications. The very fundamental goals are ensuring storage integrity and auditability, confidentiality, and access pattern hiding, as well as combinations of all of them. Despite sharing a common setting, security analyses of these tasks are often performed in a stand-alone fashion expressed in different models, which makes it hard to assess the overall security of a protocol or application involving several security schemes at once. We fill this gap by providing a framework suitable to capture various aspects of outsourced storage security and its applications. Again, we propose the relevant idealizations and the corresponding constructions that achieve these idealizations.

Aside of the new model, Chapter 6 makes three specific contributions to the realm of outsourced storage security:

1. We present a novel definition for secure and robust outsourcing schemes and underline why this is needed in practice. Our definition is stronger than previous definitions for oblivious RAM or software protection in that it assures strong security guarantees against active attacks. Schemes meeting the definition not only assure that an attacker cannot learn the access pattern, but guarantee resilience to errors and the prevention of targeted attacks to specific locations. Unfortunately, several existing schemes cannot achieve this high level of security. For completeness, we provide a protocol based on Path ORAM that showcases that stronger security is actually achievable.

2. We present a novel definition for auditable storage, capturing the guarantee that a successful audit implies that the current server state allows the client to retrieve his data. We develop an audit mechanism, based on secure and robust outsourcing schemes, that is similar to the construction by Cash et al. (Eurocrpyt 2013), but is universally composable and fault-tolerant.

3. We revisit the security claim of a widely-used challenge-response audit mechanism, in which the server has to compute a hash $H(F||c)$ on the file F concatenated with a uniformly random challenge c chosen by the client. Being concerned with composable security, we prove that this audit mechanism is not secure, even in the random

oracle model, without additional assumptions. The composable security of this basic audit scheme was implicitly assumed in Ristenpart et al. (Eurocrypt 2011). To complete the picture, we state the additional assumptions for this audit mechanism to be provably secure and investigate the (in)applicability of hash-function constructions in this setting.

The results of Chapter 6 are published in [BM18].

1.3.3 Digital Signatures Schemes

The stage for Chapter 7 are digital signature schemes (DSS), a very fundamental cryptographic primitive with various and diverse application scenarios. Because of this, a security definition that specifies the guarantees of a DSS under composition is needed. Canetti (FOCS 2001, CSFW 2004) as well as Backes, Pfitzmann, and Waidner (CCS 2003) have described ideal functionalities for signatures in their respective composable-security frameworks. All these proposals have their benefits and shortcomings, one reason being that due to this diversity in applications, it is not at all clear for signatures how an idealization should be formulated best.

We therefore require a more fundamental and more abstract approach to capture what signature schemes should ideally achieve. In Chapter 7 we hence describe digital signature schemes from a different, more abstract perspective. Instead of modeling all aspects of a DSS in a monolithic ideal functionality, our approach characterizes a DSS as a construction of a repository for authentically reading values written by a certain party from certain assumed repositories, e.g., for transmitting verification key and signature values. This approach resolves several technical complications of previous simulation-based approaches, captures the security of signature schemes in an abstract way, and allows for modular proofs. One major contribution we make is that our new definition is equivalent to the existence of a signature scheme that is existentially unforgeable under chosen message attacks. We hope that our more fundamental approach can be the foundation for studying idealizations of cryptographic primitives for which a main or most important application scenario cannot be properly identified.

The results of Chapter 7 are published in [BMT18].

1.3.4 Blockchain Protocols

In 2008, an unknown person (or group of people) referred to by the pseudonym Satoshi Nakamoto first proposed Bitcoin as a decentralized cryptocurrency. After this whitepaper has been published, the area of cryptocurrencies, and more general blockchain protocols, have received a lot of attention, and researchers have been analyzing and/or predicting its behavior under different attack scenarios. However, a core question remained, namely how we can idealize the service that Bitcoin provides, or more explicitly, what functionality can Bitcoin provide to cryptographic protocols and what assumptions are needed to realize such a functionality?

An intuitive answer to this question was already given in Nakamoto's original work: Bitcoin aims to achieve some form of consensus on a set of valid transactions. The core difference of this consensus mechanism with traditional consensus is that it does not rely on having a known (permissioned) set of participants, but everyone can join and leave at any point in time. This is often referred to as the *permissionless* model. Consensus in this model is achieved by shifting from the traditional assumptions on the fraction of cheating versus honest participants, to assumptions on the collective computing power of the cheating participants compared to the total computing power of the parties that support the consensus mechanism. The core idea is that in order for a party's action to affect the system's behavior, it needs to prove that it is investing sufficient computing resources. In Bitcoin, these resources are measured by means of solutions to a presumably computation-intensive problem.

Although the above idea is implicit in Nakamoto's paper, a formal description of Bitcoin's goal had not been proposed or known to be achieved (and under what assumptions) until the recent works of Garay, Kiayias, and Leonardos (Eurocrypt 2015) and Pass, Seeman, and shelat (Eurocrypt 2016). However, these existing security analyses are property-based, and as such they do not support composition.

In Chapter 8, we put forth a universally composable treatment of the Bitcoin protocol. We specify the goal that Bitcoin aims to achieve as a ledger functionality in the (G)UC model of Canetti et al. [TCC'07]. We prove that the functionality we proposed here is securely UC realized under standard assumptions by an appropriate abstraction of Bitcoin as a UC protocol. We further show how known property-based approaches can be cast as special instances of our treatment and how their underlying

assumptions can be cast in (G)UC without restricting the environment or the adversary.

The results of Chapter 8 are published in [BMTZ17] which serves as the foundation of further research in the blockchain area [BGM+18, BGK+18].

1.4 Related Work

This thesis is divided into several parts, where each part covers a certain field. Due to this variety, we provide the relevant related work for each of the topics in the respective chapters. The bibliography is found at the end of this document.

Chapter 2

Preliminaries

2.1 Notation

2.1.1 General conventions

We describe algorithms and the behavior of systems with pseudocode using the following conventions: We write $x \leftarrow y$ for assigning the value y to the variable x. For a distribution $\mathcal{D}$ over some set, $x \twoheadleftarrow \mathcal{D}$ denotes sampling x according to $\mathcal{D}$. For a finite set X, $x \twoheadleftarrow X$ denotes assigning to x a uniformly random value in X. For a positive integer $n \in \mathbb{N}$, we define $[n] := \{1, \ldots, n\}$.

We denote the empty list by $[\,]$ and for a list L, $L \parallel x$ denotes the list L with x appended. Furthermore, $|L|$ denotes the number of elements in L and the ith element in L is denoted by $L[i]$ for $i \in \{1, \ldots, |L|\}$. For a FIFO queue $\mathcal{Q}$, we write $\mathcal{Q}.\texttt{enqueue}(x)$ to insert x into the queue and $\mathcal{Q}.\texttt{dequeue}()$ to retrieve (and remove) the element of the queue that was inserted first among all remaining elements. For $n, m \in \mathbb{N}$, $\mathrm{Inj}\,(\Sigma^n, \Sigma^m)$ denotes the set of injective functions $\Sigma^n \to \Sigma^m$. For an injective function $f\colon X \to Y$, we denote by f^{-1} the function $Y \to X \cup \{\perp\}$ that maps y to the preimage of y under f if existing, and to the distinct element $\perp$ otherwise.

Typically queries to systems (for example a channel) consist of a suggestive keyword and a list of arguments, such as $(\texttt{send}, m)$ to send a message m. We ignore keywords in writing the domains of arguments, e.g.,

$(\texttt{send}, m) \in \mathcal{M}$ indicates that $m \in \mathcal{M}$. The systems generate a return value upon each query which is output at an interface of the system. We omit writing return statements in case the output is a simple constant whose only purpose is to indicate the completion of an operation.

2.1.2 Algorithms, Games, and Random Experiments

When describing game-based security, we stick to the following conventions. If $\mathcal{A}$ is an algorithm then $y \leftarrow \mathcal{A}(x)$ denotes executing $\mathcal{A}$ on input x and assigning the return value to y. If $\mathcal{A}$ is probabilistic, then $\mathcal{A}(x; r)$ denotes the execution of $\mathcal{A}$ on input x and randomness $r \in \{0,1\}^*$ of sufficient length, and in the expression $\mathcal{A}(x)$, the randomness is chosen uniformly at random. For algorithms $\mathcal{A}$ and $\mathcal{O}$, $\mathcal{A}^{\mathcal{O}(\cdot)}(x)$ denotes the execution of $\mathcal{A}$ (on input x) with oracle access to $\mathcal{O}$. We often call $\mathcal{O}$ simply an oracle. For a cryptographic game G that typically consists of a set of oracles provided to an adversary $\mathcal{A}$ (such as an encryption or decryption oracle), we write $\mathcal{A}^{\mathsf{G}}(\cdot)$ to denote the execution of $\mathcal{A}$ having oracle access to the oracles provided by G. Often, the input x to the adversary is the encoding of a security parameter in such cases we omit it from the description for simplicity (see below). When talking about game-based security, this execution is called the random experiment of an adversary $\mathcal{A}$ with game G. For such random experiments, we use the standard notation $\Pr\left[\mathcal{A}^{\mathsf{G}} = 1\right]$ to denote the probability that the output of an adversary is 1 when interacting with game G, and $\Pr\left[\mathcal{A}^{\mathsf{G}} \text{ sets } \mathsf{flag}\right]$ to denote the probability that the adversary sets a boolean (or binary) variable flag, (defined by the security game G), to the value true (or 1). For an arbitrary random experiment Exp and an event E defined with respect to that experiment, we use the notation $\Pr^{\mathsf{Exp}}[E]$ to denote the probability of event E.

To measure the performance of an adversary $\mathcal{A}$, one usually defines an advantage function which is a function of a security parameter κ. To simplify notation, we often omit the security parameter as an explicit input, for example we just write $\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}\mathcal{A}}$ for the CPA-advantage of an adversary $\mathcal{A}$ (with respect to a scheme $\mathcal{E}$) instead of $\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}\mathcal{A}}(\kappa)$.

In this thesis, we follow a concrete security approach, i.e., asymptotic considerations and notions such as efficiency and negligibility do not play a major role. For completeness, whenever we refer to an efficient algorithm

$\mathcal{A}$ in explanations or definitions, we mean probabilistic polynomial time, i.e., there is a polynomial $p(\cdot)$ such that the execution $\mathcal{A}(x)$ terminates after at most $p(|x|)$ steps in a standard model of computation. In a game-based cryptographic treatment, an adversary $\mathcal{A}$ typically takes as input the unary encoding 1^κ of the security parameter κ and is required to be efficient with respect to this input. Finally, a function f is called negligible if it vanishes faster than the inverse of any polynomial, i.e., for any polynomial $q(\cdot)$ there is an integer $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, $f(n) < \frac{1}{q(n)}$.

2.2 Basic Cryptographic Primitives

In this section, we define those cryptographic primitives that are used throughout several chapters of this work. More specialized primitives, such as authenticated encryption or signcryption, are presented in the respective chapters.

2.2.1 Symmetric-Key Encryption

In this work, we need symmetric-key encryption schemes secure against chosen-plaintext attacks.

Definition 2.2.1. A symmetric-key (or private-key) encryption scheme $\mathcal{E} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for message space $\mathcal{M}$, key space $\mathcal{K}$, and ciphertext space $\mathcal{C}$ consists of a (probabilistic) key generation algorithm Gen that returns a key $sk \in \mathcal{K}$, a (probabilistic) encryption algorithm Enc that given a message $m \in \mathcal{M}$ and the key $sk \in \mathcal{K}$ returns a ciphertext $c \leftarrow \mathsf{Enc}(sk, m) \in \mathcal{C}$, and a (possibly probabilistic) decryption algorithm Dec, that given a ciphertext $c \in \mathcal{C}$ and the key $sk \in \mathcal{K}$ returns a value $m' \leftarrow \mathsf{Dec}(sk, c) \in \mathcal{M} \cup \{\perp\}$. We require correctness, i.e., that $\mathsf{Dec}(sk, \mathsf{Enc}(sk, m)) = m$ for all keys sk in the support of Gen and all messages $m \in \mathcal{M}$.

Definition 2.2.2. For a symmetric-key encryption scheme $\mathcal{E}$, we define the CPA-advantage of an adversary $\mathcal{A}$ as the difference in the probability that it outputs 1 when interacting with game $\mathsf{IND\text{-}CPA}^0_{\mathcal{E}}$ or with game $\mathsf{IND\text{-}CPA}^1_{\mathcal{E}}$ depicted in Figure 2.1, i.e.,

$$\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E},\mathcal{A}} := \Pr\left[\mathcal{A}^{\mathsf{IND\text{-}CPA}^0_{\mathcal{E}}} = 1\right] - \Pr\left[\mathcal{A}^{\mathsf{IND\text{-}CPA}^1_{\mathcal{E}}} = 1\right].$$

The scheme $\mathcal{E}$ is called IND-CPA-secure if this advantage is negligible for all efficient adversaries.

2.2.2 Message Authentication Codes

In this work, we use the following simple definition of a message authentication code.

Definition 2.2.3. A message authentication code for message space $\mathcal{M}$, tag space $\mathcal{T}$, and key space $\mathcal{K}$ (with an associated distribution) is a function $f : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ (called the MAC function).

Definition 2.2.4. For a MAC function f, we define the EU-CMA advantage of an adversary $\mathcal{A}$ as the probability that it wins the unforgeability game $\mathsf{EU\text{-}CMA}_f^{\mathrm{mac}}$ depicted in Figure 2.1, i.e.,

$$\mathsf{Adv}_{f,\mathcal{A}}^{\mathsf{eu\text{-}cma}} := \Pr\left[\mathcal{A}^{\mathsf{EU\text{-}CMA}_f^{\mathrm{mac}}} \text{ sets } \mathsf{win}\right].$$

The MAC function f is called secure if this advantage is negligible for all efficient adversaries.

2.2.3 Erasure Codes

Erasure codes are a special type of error-correcting codes and defined as follows.

Definition 2.2.5. An (n, k, d) erasure code over an alphabet Σ with error symbol $\perp \notin \Sigma$, is a pair of (efficient) algorithms (E, D) that satisfy the following requirement for all $F \in \Sigma^k$: Let $\bar{F} := \mathsf{E}(F) \in \Sigma^n$ and define the set

$$\mathcal{C}_{\bar{F}} := \{\bar{F}' \in (\Sigma \cup \{\perp\})^n \mid \forall i : \bar{F}'_i \in \{\bar{F}_i, \perp\} \wedge \text{at most } d-1 \text{ positions of } \bar{F}' \text{ are equal to } \perp\}.$$

Then, for all $\bar{F}' \in \mathcal{C}_{\bar{F}}$, it holds that $\mathsf{D}(\bar{F}') = F$.

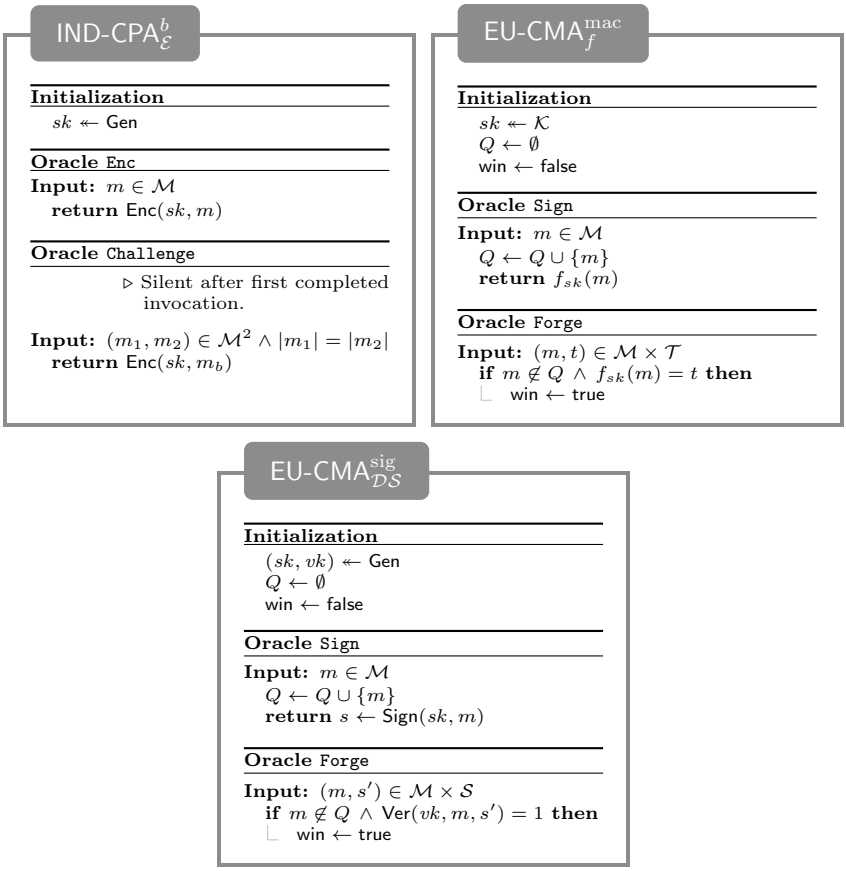


Figure 2.1: The security games to define the standard security notions that are used throughout this work.

2.2.4 Digital Signature Scheme

Signature schemes play an important role in this work and we state here the standard notion, namely existential unforgeability under chosen-message attacks.

Definition 2.2.6. A digital signature scheme $\mathcal{DS} := (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ for a message space $\mathcal{M}$, signature space $\mathcal{S}$, and key space $\mathcal{K} = \mathcal{SK} \times \mathcal{PK}$ consists of a (probabilistic) key generation algorithm Gen that returns a key pair $(sk, vk) \in \mathcal{K}$, a (possibly probabilistic) signing algorithm Sign, that given a message $m \in \mathcal{M}$ and the signing key $sk \in \mathcal{SK}$ returns a signature $s \leftarrow \mathsf{Sign}(sk, m)$, and a (possibly probabilistic, but usually deterministic) verification algorithm Ver, that given a message $m \in \mathcal{M}$, a candidate signature $s' \in \mathcal{S}$, and the verification key $vk \in \mathcal{PK}$ returns a bit $\mathsf{Ver}(vk, m, s')$. The bit 1 is interpreted as a successful verification and 0 as a failed verification. We require correctness, that is, we demand that $\mathsf{Ver}(vk, m, \mathsf{Sign}(sk, m)) = 1$ for all $m \in \mathcal{M}$ and all pairs (vk, sk) in the support of Gen.

Definition 2.2.7. For a digital signature scheme $\mathcal{DS}$, we define the EU-CMA advantage of an adversary $\mathcal{A}$ as the probability that it wins the unforgeability game $\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}$ depicted in Figure 2.1, i.e.,

$$\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathcal{DS},\mathcal{A}} := \Pr\left[\mathcal{A}^{\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}} \text{ sets } \mathsf{win}\right].$$

The scheme $\mathcal{E}$ is called EU-CMA-secure if this advantage is negligible for all efficient adversaries.

2.3 Constructive Cryptography

In this section, we outline the model that underlies most of the results stated in this work. While constructive cryptography is a general theory that goes beyond cryptography, the purpose of this section is to describe the way we use it to prove our cryptographic statements.

2.3.1 Basic Concepts

Discrete Systems

The basic objects in the constructive security statements of this work are reactive discrete systems that can be queried by their environment: Each interaction consists of an input from the environment and an output that is given by the system in response. Discrete reactive systems are modeled formally by random systems [Mau02], and an important similarity measure on those is given by the distinguishing advantage. More formally, the advantage of a distinguisher $\mathbf{D}$ in distinguishing two discrete systems, say $\mathbf{R}$ and $\mathbf{S}$, is defined as

$$\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{S}) \;=\; \Pr[\mathbf{DR} = 1] - \Pr[\mathbf{DS} = 1]\,, \tag{2.1}$$

where $\Pr[\mathbf{DR} = 1]$ denotes the probability that $\mathbf{D}$ outputs 1 when connected to the system $\mathbf{R}$. More concretely, $\mathbf{DR}$ is a random experiment, where the distinguisher repeatedly provides an input to one of the interfaces and observes the output generated in reaction to that input before it decides on its output bit.

A further important concept for discrete systems is a *monotone binary output (MBO)* [Mau02] or *bad event* [BR06]. This concept is used to define a similarity between two systems, the *game equivalence* [Mau02] or *equivalence until bad* [BR06], which means that two systems behave equivalently until the MBO is set (i.e., as long as the bad event does not occur), but may deviate arbitrarily thereafter. A widely-used result is the so-called *Fundamental Lemma of Game Playing* [Mau02, BR06], which states that the distinguishing advantage between two such systems is bounded by the probability of provoking the MBO (i.e., bad event).

Resources and Converters

The central object in constructive cryptography is that of a resource available to parties, and the resources we discuss in this work are modeled by reactive discrete systems. As in general the same resource may be accessible to multiple parties, such as a communication channel that allows a sender to input a message and a receiver to read it, we assign inputs to certain *interfaces* that correspond to the parties: the sender's interface allows to input a message to the channel, and the receiver's

interface allows to read what is in the channel. More generally, a resource is a discrete system with a finite set of interfaces $\mathcal{I}$ via which the resource interacts with its environment.

Converters model protocols used by parties and can attach to an interface of a resource to change the inputs and outputs at that interface. This composition, which for a converter π, interface I, and resource $\mathbf{R}$ is denoted by $\pi^I\mathbf{R}$, again yields a resource. In this work, a converter π is modeled as a system with two interfaces: the *inner interface* `in` and the outer interface `out`. The inner interface can be connected to an interface I of a resource $\mathbf{R}$ and the outer interface then becomes the new interface I of resource $\pi^I\mathbf{R}$. Two special converters in this work are the identity converter `1`, which does not change the behavior at an interface, and the converter `0`, which blocks all interaction at an interface (no inputs or outputs). For two converters π_1, π_2, their concatenation $\pi_1 \circ \pi_2$ (sometimes called sequential composition) is a converter defined via $(\pi_1 \circ \pi_2)^I\mathbf{R} := \pi_1^I\pi_2^I\mathbf{R}$.

For a tuple of converters $\pi = (\pi_{I_1}, \ldots, \pi_{I_n})$ with $I_i \in \mathcal{I}$, and a subset of interfaces $\mathcal{P} \subseteq \{I_1, \ldots, I_n\}$, $\pi_{\mathcal{P}}\mathbf{R}$ denotes the resource where π_I is connected to interface I of $\mathbf{R}$ for every $I \in \mathcal{P}$. In this context, we also use the convention that $\overline{\mathcal{P}} := \mathcal{I} \setminus \mathcal{P}$. Finally, the concatenation of two tuples of converters (of the same size) is simply defined as the tuple obtained by component-wise concatenation of the respective converters.

For $\mathcal{I}$-resources $\mathbf{R}_1, \ldots \mathbf{R}_m$ the *parallel composition* $[\mathbf{R}_1, \ldots, \mathbf{R}_m]$ is again an $\mathcal{I}$-resource that provides at each interface access to the corresponding interfaces of all subsystems. If the two interfaces do not have the same interface set, one can simply add "dummy interfaces" that do not take any input and output to lift them to identical interface sets. In this sense, we also allow resources with different interface sets here. Finally, for converters $\pi_1, \ldots, \pi_m$, their parallel composition $[\pi_1, \ldots, \pi_m]$ is a converter defined via $[\pi_1, \ldots, \pi_m]^I[\mathbf{R}_1, \ldots, \mathbf{R}_m] := [\pi_1^I\mathbf{R}_1, \ldots \pi_m^I\mathbf{R}_m]$. Similar to above, the parallel composition of tuples of converters (of the same size) is the tuple obtained by component-wise parallel composition of the respective converters.

Protocols and Types of Interfaces

A protocol corresponds to the actions that parties are expected to execute, for example applying an encryption scheme to protect the content of

a message). For each interface, these actions are formally captured by attaching a particular converter. Since in a cryptographic context some parties may deviate from the prescribed behavior, we naturally have several types of interfaces. These types are implicit in the definition of a construction below and explained here.

- *Honest interfaces:* Interfaces that belong to this type always apply the specified converter in a construction and hence the prescribed behavior is the actual behavior. In secure communication, we often consider how the sender Alice (interface `A`) and receiver Bob (interface `B`) protect their communication against an entity controlling the network, often denoted to by the name Eve.

- *Potentially dishonest interfaces:* Interfaces of this type formalize the possibility that the behavior can deviate from what is prescribed by the protocol converter for this interface. Such dishonest behavior is thus captured by replacing the protocol converter by an arbitrary, adversarial strategy. We think of the protocol converter for this interface as the default behavior in case it is honest. The set of possibly dishonest interfaces is typically called $\mathcal{U}$. Following up the basic example from above, the interface of the possibly dishonest entity that controls the network, i.e., Eve, belongs to this set (interface `E`).

- *Free interface:* A free interface models that the resource may be affected by its environment in ways we cannot or do not want to fix in the resource description. This allows to abstract from certain technical details by delegating them to the environment. As we see below, in the construction statements no converter is attached at the free interface which gives the distinguisher direct access in both, the "real world" and the "ideal world". A construction is thus valid independently of what happens at the world interface. In this thesis, a resource has at most one free interface that we call interface `W` and we refer to it as the world interface. This interface plays a crucial role in several of our results, notably in Chapter 6.

Formally, a protocol is a tuple of converters that specifies one converter for each interface except the free interface.

Filtered resources

Typically, one would like to specify that certain capabilities at an interface are only potentially available (e.g., to an attacker), but not guaranteed to be available (i.e, not a feature of a protocol). A typical example is that the leakage to the network attacker of a secure channel at interface $\mathtt{E}$ is at most the length of the message $|m|$ (potentially available), but of course not guaranteed (there exist encryption schemes that hide the length of the message). To model such situation, constructive cryptography offers the concept called *filtered resources*. Let $\mathbf{R}$ be a resource and $\phi = (\phi_{I_1}, \ldots, \phi_{I_n})$ be a vector of converters. Then, the filtered resource $\mathbf{R}_\phi$ is a $\mathcal{I}$-resource, where for an honest party at interface I_j, the interaction through the converter ϕ_{I_j} is guaranteed to be available, while interactions with $\mathbf{R}$ directly is only potentially available to dishonest parties. The converter ϕ_{I_j} can be thought of as filtering or shielding certain capabilities of interface I_j of system $\mathbf{R}$, we hence denote ϕ as the filter. In this context, $\mathbf{R}$ is called an unfiltered resource (and its behavior is equivalent to resource $\mathbf{R}_\phi$ if ϕ consists of only identity converters). Note that no filter need to be specified for the free interface. We refer the reader to [MR11] for more details and briefly mention that this concept has turned out to be useful in modeling cryptographic problems and is crucially used in Chapter 5 of this work.

2.3.2 Constructions

A constructive security statement specifies the goal of a protocol in terms of *assumed* and *constructed* resources. The goal of a protocol is to construct the ideal resource from the given ones. We directly state the central definition of a construction of [MR11] that is used throughout this work.

Definition 2.3.1. Let $\mathbf{R}_\phi$ and $\mathbf{S}_\psi$ be filtered resources with interface set $\mathcal{I}' := \mathcal{I} \cup \{\mathtt{W}\}$, $\mathtt{W} \notin \mathcal{I}$. Let $\pi = (\pi_{I_1}, \ldots, \pi_{I_{|\mathcal{I}|}})$ be a protocol. Let further be $\mathcal{U} \subseteq \mathcal{I}$ be the set of interfaces with potentially dishonest behavior and let ε be a function that maps distinguishers to a value in $[-1, 1]$. The protocol π constructs $\mathbf{S}_\psi$ from $\mathbf{R}_\phi$ within ε and with respect to potentially dishonest $\mathcal{U}$, denoted by

$$\mathbf{R}_\phi \quad \overset{(\pi, \varepsilon, \mathcal{U})}{\Longrightarrow} \quad \mathbf{S}_\psi,$$

if there exist converters $\sigma = (\sigma_{U_1}, \ldots, \sigma_{U_{|\mathcal{U}|}})$, $U_i \in \mathcal{U}$, such that for all (dishonest) subsets $\mathcal{C} \subseteq \mathcal{U}$ we have that

$$\Delta^{\mathbf{D}}(\pi_{\overline{\mathcal{C}}}\, \phi_{\overline{\mathcal{C}}}\mathbf{R},\, \sigma_{\mathcal{C}}\, \psi_{\overline{\mathcal{C}}}\mathbf{S}) \leq \varepsilon(\mathbf{D}) \tag{2.2}$$

for any distinguisher $\mathbf{D}$.

Since in this work we state concrete security statements and reductions, the mapping $\varepsilon(\cdot)$ is typically the advantage of an adversary $\mathcal{A} = \rho(\mathbf{D})$ in a related security game (such as the forgery game) where $\rho(\cdot)$ stands for a mapping from distinguishers to adversaries, for example implemented by a black-box construction of such an adversary $\mathcal{A}$ from a distinguisher $\mathbf{D}$. To give some intuition, the condition in Definition 2.3.1 ensures that for any combination of dishonest interfaces, whatever they can do in the assumed resource using the unfiltered capabilities, they could do as well with the constructed resource by applying the *simulators* σ_{U_i} to the respective (unfiltered) interfaces U_i of the ideal resource. Turned around, if the constructed resource is secure by definition (for example, a secure channel does potentially leak at most the length of a message), there is no successful attack on the protocol.

2.3.3 Composition

The notion of construction is composable, which intuitively means that the constructed resource can be replaced in any context by the assumed resource with the protocol attached. We refer to [MR11, Mau12] for the details and the proof and provide here the statement for completeness.

Theorem 2.3.2 (Composition Theorem)**.** *Let $\mathbf{R}_\phi$, $\mathbf{R}'_{\phi'}$, $\mathbf{S}_\psi$, $\mathbf{S}'_{\psi'}$, and $\mathbf{T}_\tau$ be (filtered) resources and let π_1, π_2, π'_1 be protocols as in Definition 2.3.1. Let $\pi_3 \coloneqq \pi_2 \circ \pi_1$ and $\pi'_3 \coloneqq [\pi_1, \pi'_1]$ be the sequential and parallel compositions of these protocols. We have*

$$\mathbf{R}_\phi \overset{(\pi_1,\varepsilon_1,\mathcal{U})}{\Longrightarrow} \mathbf{S}_\psi \;\wedge\; \mathbf{S}_\psi \overset{(\pi_2,\varepsilon_2,\mathcal{U})}{\Longrightarrow} \mathbf{T}_\tau \;\Longrightarrow\; \mathbf{R}_\phi \overset{(\pi_3,\varepsilon_3,\mathcal{U})}{\Longrightarrow} \mathbf{T}_\tau$$

and

$$\mathbf{R}_\phi \overset{(\pi_1,\varepsilon_1,\mathcal{U})}{\Longrightarrow} \mathbf{S}_\psi \wedge \mathbf{R}'_{\phi'} \overset{(\pi'_1,\varepsilon'_2,\mathcal{U})}{\Longrightarrow} \mathbf{S}'_{\psi'} \;\Longrightarrow\; [\mathbf{R}_\phi, \mathbf{R}'_{\phi'}] \overset{(\pi'_3,\varepsilon'_3,\mathcal{U})}{\Longrightarrow} [\mathbf{S}_\psi, \mathbf{S}'_{\psi'}],$$

for specific performance functions $\varepsilon_3(\cdot)$ and $\varepsilon'_3(\cdot)$. In particular, considering only the universe of efficient systems, it holds that if $\epsilon_1(\cdot)$, $\epsilon_2(\cdot)$, and $\epsilon'_2(\cdot)$ are negligible (for all efficient distinguishers), then the performance functions $\epsilon_3(\cdot)$ and $\epsilon'_3(\cdot)$ are negligible (for all efficient distinguishers).

The last part of the theorem is stated since it is the most frequent formulation found in the literature and corresponds to the intuition that the constructed resource can be replaced in any context by the cryptographic construction that achieves it — without affecting any of the security guarantees. We do not state a definition of efficient systems here, but simply note that what we expect from such a notion is that efficiency is preserved when composing systems [MR11]. We omit the details here (including the exact reductions involved in the proof of Theorem 2.3.2) because they are not important to follow the results in this thesis.

2.3.4 An Important Special Case

In many cases, there is only one single interface with possibly dishonest behavior and this interface is typically denoted by $\mathtt{E}$. Such constructions often occur in practice, namely in settings where the goal of honest parties is to protect themselves against one central adversary that tries to break the security they want to achieve. A prominent example is protecting communication between two parties (the so-called Alice-Bob-Eve setting), or in secure storage protocols where the possibly dishonest entity is the server.

In such cases, it is often simpler to directly define the involved resources with the worst-case capabilities guaranteed at interface $\mathtt{E}$. We now discuss how Definition 2.3.1 simplifies substantially in this case and present a simpler definition that Definition 2.3.1 for the case that $\mathcal{U} = \{\mathtt{E}\}$ as we explain below. The main idea is that we do not invoke the concept of filtered resources and instead explicitly speak of the default behavior at interface $\mathtt{E}$ (when no attacker is present). Furthermore, a protocol needs only specify the converters that are attached at the honest interfaces.

In the following, let $\mathcal{P} := \{\mathtt{P}_1, \ldots, \mathtt{P}_k\}$ denote the set of interfaces with honest behavior. The simplified definition reads as follows:

Definition 2.3.3. Let $\mathbf{R}$ and $\mathbf{S}$ be resources with interface set $\mathcal{I} = \mathcal{P} \cup \{\mathtt{E}, \mathtt{W}\}$. Let ε be a function that maps distinguishers to a value in $[0, 1]$ and let sim be a converter (the simulator). Let $\pi = (\pi_{\mathtt{P}_1}, \ldots, \pi_{\mathtt{P}_k})$

be the converters attached at the honest interfaces and let $\mathsf{noAtck}_{\mathbf{R}}$ and $\mathsf{noAtck}_{\mathbf{S}}$ be converters that describe the default behavior at interface $\mathtt{E}$. The protocol π constructs resource $\mathbf{S}$ from resource $\mathbf{R}$ (with potentially dishonest $\mathtt{E}$) within ε and with respect to the simulator sim and the pair $(\mathsf{noAtck}_{\mathbf{R}}, \mathsf{noAtck}_{\mathbf{S}})$, if for all distinguishers $\mathbf{D}$,

$$\Delta^{\mathbf{D}}(\mathsf{noAtck}_{\mathbf{R}}^{\mathtt{E}}\,\pi_{\mathcal{P}}\mathbf{R}, \mathsf{noAtck}_{\mathbf{S}}^{\mathtt{E}}\,\mathbf{S}) \leq \varepsilon(\mathbf{D}) \qquad \text{(Correctness)}$$

$$\Delta^{\mathbf{D}}(\pi_{\mathcal{P}}\mathbf{R},\ \mathsf{sim}^{\mathtt{E}}\,\mathbf{S}) \leq \varepsilon(\mathbf{D}). \qquad \text{(Security)}$$

The first condition ensures that the protocol implements the required functionality if there is no attacker. For example, for communication channels, all sent messages have to be delivered when no attacker interferes with the protocol. The second condition ensures that whatever Eve can do with the assumed resource, she could do as well with the constructed resource by using the simulator sim.

It is easy to see that whenever a protocol constructs resource $\mathbf{S}$ from resource $\mathbf{R}$ according to Definition 2.3.3 this corresponds to a construction of resource $\mathbf{S}_{\psi}$ from resource $\mathbf{R}_{\phi}$ according to Definition 2.3.1, where interface $\mathtt{E}$ is filtered by converters $\mathsf{noAtck}_{\mathbf{S}}$ and $\mathsf{noAtck}_{\mathbf{R}}$, and interfaces $\mathtt{P}_k$ are not filtered (i.e, the filter is the identity converter). The protocol π' that achieves this construction according to Definition 2.3.1 is defined by $\pi' \coloneqq (\pi_{\mathtt{P}_1}, \ldots, \pi_{\mathtt{P}_k}, 1)$. Note that the definition also applies to resources without free interface, in the sense that it can be considered as inactive.

2.3.5 An Example of Resources and Constructions

In the literature, communication channels are modeled as resources with three interfaces: Interface $\mathtt{A}$ for sender Alice, interface $\mathtt{B}$ for receiver Bob, and interface $\mathtt{E}$ for adversary Eve. Different types of such channels have been studied, which differ in the capabilities of the adversary Eve, and how to construct such channels [CMT13, Mau12, MRT12]. We will briefly discuss the structure of these statements to provide a concrete example that relates to the terms of assumed and constructed resources. Throughout, we assume a generic message domain of Σ^* for some alphabet (i.e., a finite non-empty set) Σ.

Assumed Resources

Insecure channel. The insecure channel **IC** allows messages to be input repeatedly at interface `A`. Each message is subsequently leaked at the `E`-interface. At interface `E`, arbitrary messages (including those that were previously input at interface `A`) can be injected such that they are delivered to `B`. This channel does not give any security guarantees to Alice and Bob. A formal description is provided in Figure 2.2.

Shared Secret Key. The shared secret key $\mathbf{SK}_{\mathcal{K}}$ samples a key k according to the key distribution $\mathcal{K}$ and upon queried at either interface `A` or `B`, it outputs k; interface `E` remains inactive. A formal description is provided in Figure 2.2.

Constructed Resource

Secure channel. The typical formalization of a secure channel follows the same basic structure as an insecure channel but where the ability of the adversary is limited to seeing the length of the transmitted messages and to deliver messages input at interface `A`. In particular, the adversary cannot inject new messages or induce out-of-order message delivery. A description of the secure channel can be derived from Figure 2.2 by omitting the `inject`-query and by restricting the leakage at interface `E` to $|m|$ on inputs $(\texttt{send}, m)$ at interface `A`.

Secure Channel Protocol as a Construction

We can now elegantly state what a secure channel protocol should achieve: it should construct a secure channel from an insecure channel and a shared secret key, possibly based on some computational assumptions. A channel protocol would formally be specified as a pair of converters $(\mathsf{snd}, \mathsf{rcv})$ for the sender Alice and receiver Bob, respectively. We will not give a concrete protocol here and refer to [CMT13, MRT12], and in particular to Chapter 3 of this thesis, where candidate constructions are presented. These constructions are typically based on cryptographic primitives (such as authenticated-encryption schemes) that satisfy a certain game-based security notion.

The security condition of Definition 2.3.3 is illustrated graphically in Figure 2.3 and shows what we usually call the real and ideal worlds:

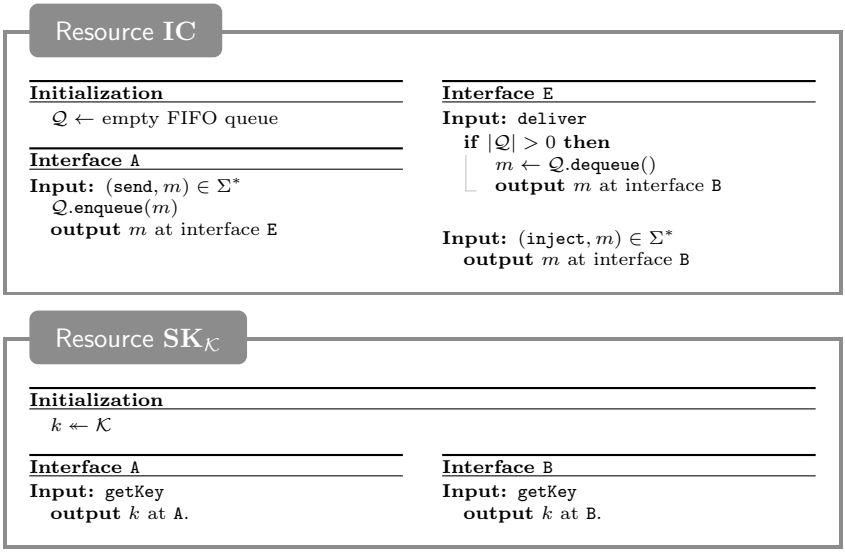


Figure 2.2: Top: The insecure channel resource; Bottom: The shared secret-key resource.

the real world consists of the assumed resources, where Alice and Bob apply their respective protocol converters, while Eve is given access to the insecure channel. Formally, this is captured by the resource $\mathsf{snd}^{\mathtt{A}}\,\mathsf{rcv}^{\mathtt{B}}\,[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]$. This world has to be proven indistinguishable from the ideal world, where Alice and Bob have access to the secure channel, and where a simulator emulates the "real-world" view at interface E (only having access to the length of transmitted messages). The ideal world is formally captured as the resource $\mathsf{sim}^{\mathtt{E}}\,\mathbf{SEC}$.

A channel protocol also has to ensure correctness, i.e., that sent messages can be correctly recovered by Bob, in particular when no attacker interferes with the protocol execution. For this, we can specify the default behavior at interface E, required by Definition 2.3.3, by the special converter dlv that is attached at interface E and always ensures the delivery of messages. Concretely, on any input at its inner interface, dlv outputs deliver to the channel connected to its inner interface and does not

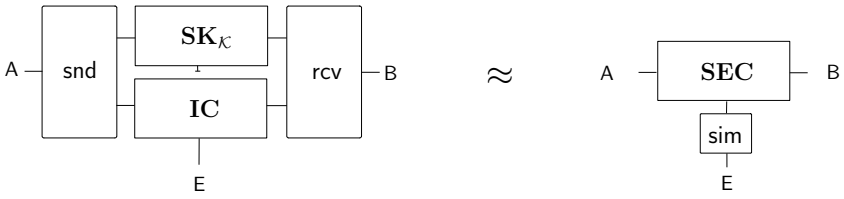


Figure 2.3: Illustration of security condition for the Alice-Bob-Eve setting to formalize secure channel protocols.

provide any service at its outer interface. The correctness condition then requires that the two systems $\mathsf{snd}^{\mathsf{A}}\,\mathsf{rcv}^{\mathsf{B}}\,\mathsf{dlv}^{\mathsf{E}}\,[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]$ and $\mathsf{dlv}^{\mathsf{E}}\,\mathbf{SEC}$ are indistinguishable. Note that the resource $\mathsf{dlv}^{\mathsf{E}}\,\mathbf{SEC}$ is actually a perfect channel from Alice to Bob.

2.3.6 Specifications

In the cryptographic literature, construction statements typically relate fully specified systems. Consider the definition Definition 2.3.3 with $\varepsilon = 0$ (perfect security) and for the Alice-Bob-Eve setting as discussed in the previous section. The security condition in this simple setting relates two fully specified systems: the "real world" $\pi_1^{\mathsf{A}}\,\pi_2^{\mathsf{B}}\,\mathbf{R}$ and the "ideal world" $\sigma^{\mathsf{E}}\mathbf{S}$. A benefit of such statements is that we know exactly what we assume and what we get. On the other hand, if the assumed resource in an application is just slightly different, the construction statement does not apply anymore. Also, the exact behavior of simulator σ is not always of primary interest, since the honest parties do anyway not have control over the adversarial strategy and hence they only care that what they get is "basically" $\mathbf{S}$.

Quite surprisingly, the idea that it could be desirable to make statements about settings that are not fully specified has not received much attention in the cryptographic literature. The approach was made formal by the concept of *specifications* introduced by Maurer and Renner [MR16]. Specifications are sets of resources that, for example, fulfill a certain property. As such they are suitable to express an incomplete description of a resource, namely by considering the set of all resources that adhere to such a (partially defined) description. Maurer and Renner describe concrete

types of specifications such as all resources that can be distinguished from a specific one by at most a certain advantage, or all resources that are obtained from a specific one by applying certain transformations. A protocol constructs from a specification $\mathcal{R}$ another specification $\mathcal{S}$ if for each system $\mathbf{R}$ that satisfies (or is contained in) $\mathcal{R}$ there exists a system $\mathbf{S}$ that satisfies $\mathcal{S}$ such that the protocol constructs $\mathbf{S}$ from $\mathbf{R}$ [MR16].

In this context, the basic example from above can be understood in a very clean manner. If we define the ideal specification to be the set $\mathcal{S}$ of resources that contains $\mathbf{S}$ and all resources that are derived from $\mathbf{S}$ by applying an arbitrary converter at interface $\mathtt{E}$ (recall that all such systems are acceptable in our view), then Definition 2.3.3 directly implies that the real world $\pi_1^{\mathtt{A}}\,\pi_2^{\mathtt{B}}\,\mathbf{R}$ satisfies specification $\mathcal{S}$, which is a very concise statement.

While statements about fully-specified settings are of interest when specifying concrete applications and what they should achieve (e.g., to understand TLS), the specification concept turns out to be very powerful when trying to understand low-level cryptographic primitives such as digital signatures that have a broad variety of applications and where it seems inherently difficult to model a "characteristic" construction statement. In this thesis, Chapter 7 is devoted to exactly this question.

2.4 Overview of the UC Framework

Some of the results in this work are formulated in the universal composability (UC) framework introduced by Canetti [Can01a]. We give a brief introduction into the main notation of this framework.

2.4.1 Basics

The goal of the UC framework is to capture what it means for a protocol to securely carry out a task. Different to constructive cryptography, UC first defines the process of executing a protocol in some environment and in the presence of an adversary, next it defines an ideal process to formalize what securely carrying out the task means, and finally one has to prove that no (efficient) environment can distinguish the real process and the ideal process. Similar to constructive cryptography, the core defining element of the ideal process is the ideal functionality, which can

be thought of as an incorruptible party. We briefly describe the main ingredients first and then describe the real and ideal process.

Protocol and protocol instances. Formally, a protocol π is an algorithm for a distributed system and formalized as an interactive Turing machine. An ITM has several tapes, for example an identity tape (read-only), an activation tape, or input/output tapes to pass values to its program and return values back to the caller (e.g., the environment). An ITM also has communication tapes that model messages sent to and received from the network.

While an ITM is a static object, UC defines the notion of an ITM instance (denoted ITI), which is defined as the pair (M, id), where M is the description of an ITM and $id = (\mathsf{sid}, \mathsf{pid})$ is the identity string consisting of a session identifier sid and a party identifier pid. Each instance is associated with a configuration, which is as usual the contents of all of its tapes and the heads, and the control state of that ITM.

An instance, also called a session, of a protocol π (represented as an ITM M_π) with respect to a session number sid is defined as a set of ITIs (M_π, id_i) with $id_i = (\mathsf{pid}_i, \mathsf{sid})$.

Network and adversary. The UC model does not give any guarantee for its built-in network. The network is asynchronous without guaranteed delivery, the messages are visible by an adversary and there is no authenticity guarantee on the content or originator of a message.

The adversary $\mathcal{A}$ is also defined as an ITM. Aside of its capabilities to send and read messages, it can at any time issue special corruption messages to corrupt protocol ITMs. When an ITM is corrupted, the adversary does not only learn the contents of all tapes, but it can also act in the name of this ITM, meaning that whenever this ITM is activated, the adversary gets actually activated and can decide on the next steps. This corruption dynamics is the standard form of corruption and we call such an adversary active and adaptive.

There are several deviations from this corruption model. A famous model is passive security, where the adversary is as above, but cannot decide on a corrupted parties next steps. Instead, the party follows the protocol. Another common model is to restrict the adversary to static corruption, which means that an ITM can only be corrupted if the

corruption message is issued before the ITM has executed the first step of its program.

2.4.2 Real-world process

The real-world process for a protocol π is defined as follows. Let $\mathcal{Z}$ be an environment machine and let $\mathcal{A}$ denote the adversary. The execution consists of a sequence of activations, initiated by $\mathcal{Z}$, where in each activation, either $\mathcal{Z}$, $\mathcal{A}$ or some ITI running π is activated. We say that $\mathcal{Z}$ invokes a new ITI $\mathcal{Z}$ if it activates an ITI for the first time (by passing some inputs) upon which this new instance gets created (in the default configuration). All ITIs invoked by $\mathcal{Z}$ need to have unique identities, but need to have the same session-identifier (which is chosen by $\mathcal{Z}$), i.e., for all ITIs I in this execution, $id_I := (s_I, \mathsf{sid})$ for some bitstring s_I specific to this instance.

Activations and execution rules. An activated ITI can change its configuration based on its code. By the UC system model (i.e., by the definition of external-write requests), an ITI loses its activation (i.e. is forced to complete) after (1) writing a message on its communication output tape (in which case the adversary gets activated next), (1) passing an input value to a (subsidiary) ITI (like a hybrid functionality), or (3) producing an output, i.e., writing to its subroutine output tape. In cases (2) and (3) the next activated ITI is the ITI that was addressed in this external-write request.

The environment $\mathcal{Z}$ can pass inputs to and read outputs from the input/output tape of any party, respectively. The adversary $\mathcal{A}$ can access the communication tapes of the parties and deliver messages by copying the entries from an outgoing communication tape to an incoming communication tape. Following the external-write rules, if in some activation, the adversary delivers a message to an ITI, then this ITI is activated next. In addition, the adversary can corrupt parties as described above, which produces an observable special output to the environment.

The UC model follows a set of activation rules specified by the UC control function. We give here a brief overview. As already stated, the environment is activated first, and upon completion of its actions (entering a special waiting state), the adversary is activated as a second entity. The remaining execution proceeds as described above. As a convention, in

addition to the above rules, the UC execution model requires that if an ITI completes without external-write request, e.g., generating no output at all, then the environment is activated next. An important property of these rules is that they ensure uniqueness of the next activated ITI and that it allows free interaction between the adversary and the environment between any two activations of protocol ITIs.

Output and transcript. The output of the protocol execution is the output of $\mathcal{Z}$ and we assume that this output is a binary value $v \in \{0,1\}$. We denote this output by $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z,r)$ where k is the security parameter, $z \in \{0,1\}^*$ is the input to the environment, and randomness r for the entire experiment. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ denote the random variable obtained by choosing the randomness r uniformly at random and evaluating $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z,r)$. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$. By slight abuse of notation, we denote by $T_{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}}(k,z,r)$ the associated transcript of this execution, which is the concatenation of all inputs to $\mathcal{Z}$, all outputs from $\mathcal{Z}$, and all messages exchanged via the communication tapes of the ITIs. The distribution $T_{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}}(k,z)$ and ensemble $T_{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}}$ are defined analogously to above.

2.4.3 Ideal-world process

Security of protocols is defined via comparing the real-world execution with an ideal-world process that solves the task in an idealistic way. More formally, the ideal process is formulated with respect to an ITM $\mathcal{F}$ which is called an ideal functionality. In the ideal process, the environment $\mathcal{Z}$ interacts with $\mathcal{F}$, an ideal-world adversary (often called the simulator) $\mathcal{S}$ and a set of trivial, i.e., dummy ITMs representing the protocol machines. The dummy ITMs behave as follows: whenever activated with a request x, they forward the request x to $\mathcal{F}$ and output towards $\mathcal{Z}$ whatever they receive in return. $\mathcal{F}$ thereby specify all outputs generated for each party, and the amount of information the ideal-world adversary learn and what its active influence is via its interaction with $\mathcal{F}$. By definition of the corruption mechanism in standard UC, an ideal functionality is informed (via special corruption messages) which instances of the dummy ITMs are corrupted. We note that an ideal functionality itself, represented as an ITI during the protocol execution, cannot be corrupted by definition.

Based on the above definitions, the ideal-world process proceeds as the real process. It is essentially the real-world process where the ITIs running the protocol are replaced by the dummy ITIs interacting with $\mathcal{F}$ (and only one challenge session ever exists). In this interaction, the same constraints and activation sequence restrictions are enforced by the UC control function. For further details we refer to [Can01a].

We denote the output of this ideal-world process by $\text{EXEC}_{\mathcal{F},\mathcal{A},\mathcal{Z}}(k,z,r)$ where the inputs are as in the real-world process. Let $\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)$ denote the random variable obtained by choosing the randomness r uniformly at random and evaluating $\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z,r)$. Let $\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$. The transcript is defined analogously as in the real-world process and denoted $T_{\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}}(k,z,r)$.

2.4.4 Hybrid worlds

To model setup, the UC framework knows so-called hybrid worlds. We discuss two important cases of hybrid worlds that differ in whether the setup, typically called the hybrid functionality, is available only to an instance of a protocol session (standard), or to multiple protocol sessions at the same time (shared). Note that a protocol can assume several setup functionalities of both types.

Standard (local) setup

A standard setup is modeled in UC as an ideal-functionality available in a real-world protocol execution, i.e., as an incorruptible ITI $\mathcal{F}$ that provides certain ideal guarantees to this protocol session. We consider here the natural case that standard setups are available in real-world processes only. However, note that while the following conventions could be applied to ideal-world-processes as well, it still seems like an uninteresting case to consider standard setups in ideal-processes. So, formally, the $\mathcal{F}$-hybrid-world process is identical to the real-world process with the following additions: The parties can interact with an a priori unbounded number of instances of $\mathcal{F}$ by standard interaction, i.e., sending messages, passing output to them, or receiving input from them. Each copy of $\mathcal{F}$, i.e., each such incorruptible ITI, is identified via a unique session identifier sid chosen by the protocol that passes inputs to it. This in particular implies a unique identity *id* of this ITI. It is stressed that by this definition, the

environment can only access $\mathcal{F}$ via calls to parties or via the adversary but not directly.

Since a protocol makes explicit which local functionalities it assumes we omit an explicit reference in the formal expressions for simplicity. For example, we just write $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ or $T_{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}}$ to denote the output or the transcript distribution ensembles in such cases.

Shared (global) setup

We briefly elaborate on the so-called externalized UC model (EUC), which is an extension of standard UC and an important special case of what is known as the generalized UC framework (GUC) [CDPW07a]. In EUC, we allow a dedicated hybrid functionality, say $\mathcal{G}$, to be declared as shared, often also denoted to as global setup. The process is identical to the hybrid-world process as above with the following addition: The UC control function also allows this special functionality $\mathcal{G}$ to directly interact with the environment $\mathcal{Z}$ via dummy ITIs. Technically, in this hybrid-world process, the control function allows $\mathcal{Z}$ to spawn dummy ITIs (with unique identities) for the purpose of interacting with $\mathcal{G}$. Unlike standard setups, shared setups are available in ideal-world processes as well, where they can interact, subject to the usual rules, with the environment or the ideal functionality and the dummy ITIs that represent the protocol interface or the access point to the shared setup. Recall that dummy ITIs always forward inputs, either to the ideal functionality $\mathcal{F}$ (protocol inputs of this session), to the shared setup $\mathcal{G}$ (setup queries), and even between functionalities such as $\mathcal{F}$ and $\mathcal{G}$, as for example defined in [CSV16a].

We conclude that $\mathcal{G}$ can be used to model shared state across sessions, and also how other sessions can interfere with the setup. In this work, the EUC notion is the way we model global setups in UC. Notable examples of shared setups include (global) random oracles, common reference strings, or clocks. We point out that for a special class of protocols, including the one in this work, the EUC notion is sufficient to satisfy the even stronger GUC notion. We do not discuss this particular class of protocols here since it is not important to understand the results in this work. The relevant definition, relating to subroutine-respecting protocols, is given in [CDPW07a] where also the associated equivalence proof of EUC and GUC is found.

If a shared setup $\mathcal{G}$ is available in the real-world or ideal-world pro-

cesses, we usually make it explicit in the notation such as $\text{EXEC}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}$ or $\text{EXEC}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

2.4.5 Secure Realization and Composition

In a nutshell, a protocol securely realizes an ideal functionality $\mathcal{F}$ if the real-world process is indistinguishable from the ideal-world process (relative to $\mathcal{F}$). If the protocol uses setup, we technically consider the hybrid-world processes instead of the plain real-world or ideal-world processes. We directly state the definitions.

Definition 2.4.1. Let us denote by $\mathcal{X} = \{X(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ and $\mathcal{Y} = \{Y(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ two distribution ensembles over $\{0,1\}$. We say that $\mathcal{X}$ and $\mathcal{Y}$ are indistinguishable if for any $c,d \in \mathbb{N}$ there exists a $k_0 \in \mathbb{N}$ such that $|\Pr[X(k,z)=1] - \Pr[Y(k,z)=1]| < k^{-c}$ for all $k > k_0$ and all $z \in \bigcup_{\kappa\leq k^d}\{0,1\}^\kappa$. We use the shorthand notation $\mathcal{X} \approx \mathcal{Y}$ to denote two indistinguishable ensembles.

Definition 2.4.2. Let $n \in \mathbb{N}$, let $\mathcal{F}$ be an ideal functionality and let π be a protocol defined for the real-world, and which potentially makes use of some local setup functionality $\mathcal{H}$ and some global setup $\mathcal{G}$. We say that π, with access to its setup, securely realizes $\mathcal{F}$ if for any (efficient) adversary $\mathcal{A}$ there exists an (efficient) ideal-world adversary (the simulator) $\mathcal{S}$ such that for every (efficient) environment $\mathcal{Z}$ it holds that $\text{EXEC}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

In the literature, the above condition is often referred to as π securely realizing functionality $\mathcal{F}$ in the $(\mathcal{G},\mathcal{H})$-hybrid world, where the type of setup is inferred by the context.

Composition. The notion of secure realization is composable. We do not give a detailed explanation as it is not important to follow the results in this work. In a nutshell, assume first that a protocol securely realizes $\mathcal{F}$ in the $\mathcal{H}$-hybrid world, where $\mathcal{H}$ denotes a standard (local) setup functionality. Let further ρ be a protocol that securely realizes $\mathcal{F}$. Then the protocol π', where each call to $\mathcal{H}$ is replaced by an invocation of protocol ρ, securely realizes $\mathcal{F}$. We refer the interested reader to [Can01a] for the general formal statement and on the exact definition of π'. Along similar lines, a composition theorem can be proven where standard (local) hybrid functionalities are replaced by the protocols securely realizing them

in the presence of an additional shared setup [CDPW07a]. Finally, we only note in passing that one can also consider replacing shared functionalities by suitable protocols. This, however, is a very subtle issue for which we refer the interested reader to [CSV16a].

2.5 Large Deviation Bounds

We use some known results to derive large deviation bounds in our probabilistic arguments. For proofs and further discussions we refer to [DP09].

Theorem 2.5.1 (Chernoff bound)**.** *Let $X_1, \ldots, X_T$ be independent random variables with $\mathbb{E}[X_i] = p_i$ and $X_i \in [0,1]$. Let $X = \sum_{i=1}^{T} X_i$ and $\mu = \sum_{i=1}^{T} p_i = \mathbb{E}[X]$. Then, for all $\Lambda \geq 0$,*

$$\Pr[X \geq (1+\Lambda)\mu] \leq e^{-\frac{\Lambda^2}{2+\Lambda}\mu};$$
$$\Pr[X \leq (1-\Lambda)\mu] \leq e^{-\frac{\Lambda^2}{2+\Lambda}\mu}.$$

Theorem 2.5.2 (Azuma's inequality (Azuma; Hoeffding).)**.** *Let $X_0, \ldots, X_n$ be a sequence of real-valued random variables so that, for all t, $|X_{t+1} - X_t| \leq c$ for some constant c. If $\mathbb{E}[X_{t+1} \mid X_0, \ldots, X_t] \leq X_t$ for all t then for every $\Lambda \geq 0$*

$$\Pr[X_n - X_0 \geq \Lambda] \leq \exp\left(-\frac{\Lambda^2}{2nc^2}\right).$$

Alternatively, if $\mathbb{E}[X_{t+1} \mid X_0, \ldots, X_t] \geq X_t$ for all t then for every $\Lambda \geq 0$

$$\Pr[X_n - X_0 \leq -\Lambda] \leq \exp\left(-\frac{\Lambda^2}{2nc^2}\right).$$

Part I

Secure Communication

Chapter 3

Authenticated Encryption

3.1 Introduction

We introduce a new abstraction of a secure channel as the idealization of authenticated encryption with additional data. We first give an overview of our new idealization and provide more background and pointers to related work on authenticated encryption.

3.1.1 Motivation and Contribution

Our new channel resource is called *augmented secure channel*, or ASC. Like most types of channels, an ASC lets a sender, Alice, send messages to a receiver, Bob. But unlike more conventional types of channels, each message has designated private and non-private parts. An active adversary, Eve, is present in the system, but its capabilities are limited to seeing the length of the private portion and the contents of the non-private portion of each message—and to entirely shutting down the channel. In particular, the adversary cannot inject messages or induce out-of-order message delivery. Additionally, the non-private portion can contain an implicit part, already known to the receiver, that is not transmitted but still authenticated, e.g., to bind the message to a given context.

The service an ASC provides is motivated by the ascendancy of both TLS and authenticated encryption. We take the rise of these tools, and what they deliver, as an indication that customary conceptualizations of

secure channels may not have been rich enough to deliver the service that protocol designers routinely need.

Authenticated encryption. While ASCs are closely related to schemes for authenticated encryption (AE) and authenticated encryption with associated data (AEAD), an ASC and an AE/AEAD-scheme are very different things. An ASC is a well-defined resource that parties can use. In contrast, an AEAD-scheme is a comparatively low-level primitive: it is a tuple of algorithms that is "good" in some particular, complexity-theoretic sense.

The AEAD notion emerged over a sequence of works [BN00, BR00, Jut01, KY01, Rog02, RBBK01, RS06] having two distinct purposes: to minimize the misuse of symmetric encryption primitives and to gain efficiency advantages over generic composition schemes (i.e., traditional ways to meld privacy-only encryption schemes and message-authentication codes). But in moving from conventional encryption to AEAD, the basic conception of what symmetric encryption *is* was thoroughly revamped: authenticity became an intrinsic part of the goal; so too did the allowance of (non-private) associated data A; while probabilism, formerly seen as indispensable, was surfaced and subsumed by a nonce N. Roughly said, an AEAD-scheme is nowadays defined as a triple of algorithms $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ where an efficient adversary $\mathcal{A}$ has poor advantage in distinguishing encryption and decryption oracles $(\mathcal{E}_k(N, A, m), \mathcal{D}_k(N, A, c))$ from oracles $(\$(N, A, m), \perp(N, A, c))$, where k is generated by $\mathcal{K}$, the $\$(N, A, m)$ oracle returns an appropriate number of random bits, the $\perp(N, A, c)$ oracle always returns $\perp$, the adversary repeats no nonce N in queries to its first oracle, and queries that would result in trivial wins are disallowed.

This new conceptualization for symmetric encryption gained rapid acceptance. The IEEE, IETF, ISO, and NIST all stepped in to standardize AEAD-schemes (e.g., in NIST SP 800-38C and SP 800-38D, IEEE 802.11i, ISO 19772, and IETF RFC 3610, 5116, 5288, 5297, and 7253) and revisions to widely-deployed protocols started to incorporate AEAD.

3.1.2 Authenticated Encryption with Associated Data

Let Σ be an alphabet. Typically an element of Σ is a bit ($\Sigma = \{0,1\}$) or a byte ($\Sigma = \{0,1\}^8$). For a string $x \in \Sigma^*$, $|x|$ denotes its length. We define

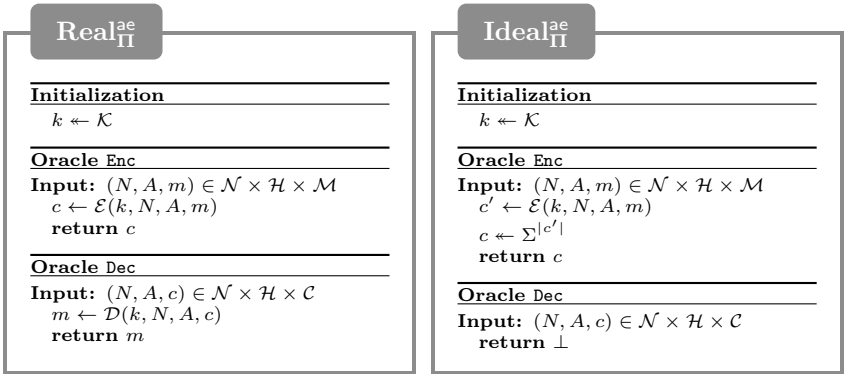


Figure 3.1: Real and ideal security game for AEAD-schemes.

the syntax of a scheme for authenticated encryption with associated data (AEAD) following [Rog02].

Definition 3.1.1. An AEAD-scheme Π is a triple of algorithms $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\mathcal{K}$ is a randomized algorithm that samples a key $k \in \Sigma^*$, $\mathcal{E}$ is a deterministic algorithm that maps a key $k \in \Sigma^*$, a nonce $N \in \mathcal{N}$, additional data[1] $A \in \mathcal{H}$, and a message $m \in \mathcal{M}$ to a ciphertext $c \in \mathcal{C}$, and $\mathcal{D}$ is a deterministic algorithm that maps $(k, N, A, c) \in \Sigma^* \times \mathcal{N} \times \mathcal{H} \times \mathcal{C}$ to $\mathcal{M} \cup \{\perp\}$. We assume the domains $\mathcal{N}$, $\mathcal{H}$, $\mathcal{M}$, and $\mathcal{C}$ are equal to Σ^* and require for all $k, N, A, m \in \Sigma^*$ that $\mathcal{D}\big(k, N, A, \mathcal{E}(k, N, A, m)\big) = m$. We further require the length of a ciphertext $|\mathcal{E}(k, N, A, m)|$ only depend on the length of the corresponding message $|m|$.

We define the security game for AEAD-schemes using the all-in-one formulation from [HKR15]. A scheme is considered secure if all valid and efficient adversaries $\mathcal{A}$ have poor advantage according to the following definition.

Definition 3.1.2. We define the advantage of an adversary $\mathcal{A}$ as the difference in the probability that it outputs 1 in the real and ideal games

[1]Regarding the notation: we use the symbol $\mathcal{H}$ (to denote the domain of the additional data) instead of the more natural choice $\mathcal{A}$ to not confuse it with our notation of an adversary. We choose $\mathcal{H}$ since we often think of the additional data as being the message header.

defined in Figure 3.1:

$$\mathsf{Adv}^{\mathsf{ae}}_{\Pi}(\mathcal{A}) \coloneqq \Pr\left[\mathcal{A}^{\mathbf{Real}^{\mathsf{ae}}_{\Pi}} = 1\right] - \Pr\left[\mathcal{A}^{\mathbf{Ideal}^{\mathsf{ae}}_{\Pi}} = 1\right].$$

An adversary is *valid* if it does not repeat `Enc` or `Dec` queries, does not ask queries $\texttt{Enc}(N, A, m)$ and $\texttt{Enc}(N, A', m')$ (i.e., does not repeat nonces), and does not ask a query $\texttt{Dec}(N, A, c)$ where c was returned by a preceding query $\texttt{Enc}(N, A, m)$.

3.1.3 The Constructive Cryptography Setting

We consider the Alice-Bob-Eve setting of constructive cryptography as introduced in Sections 2.3.4 and 2.3.5.That is, resources in this chapter have the interface set $\mathcal{I} = \mathcal{P} \cup \{\texttt{E}\}$, where the set of interfaces with honest behavior is $\mathcal{P} = \{\texttt{A}, \texttt{B}\}$.

3.2 Augmented Secure Channels

In this section, we motivate why in many applications users might need more services than provided, for example, by a secure channel as introduced in Section 2.3. We introduce a new type of channel, which we call *augmented secure channel (ASC)*, that provides those missing capabilities.

Recall that in constructive cryptography, communication channels are modeled as resources with three interfaces: Interface `A` for sender Alice, interface `B` for receiver Bob, and interface `E` for adversary Eve. Different types of such channels have been studied and we refer to Section 2.3 for a more detailed introduction.

3.2.1 An Improved Secure Channel

In many relevant security protocols, like TLS, transmitted data packets are usually divided into a header part and a payload part. While both are required to be authentic, only the payload has to remain confidential.

We further observe that the header often contains context information since binding a message to a given context is good security-engineering practice. Moreover, parts of the context are already known to the receiver. This part does not have to be transmitted but should still be authenticated. This suggests splitting the header into two parts: an explicit part and an

implicit part that describe the unknown and known parts of the header, respectively.

We conclude that there is a need for an abstract functionality that allows one to transmit a message together with the explicit part of a header such that the message remains private and the message as well as both the explicit and the implicit part of the header are authenticated.

3.2.2 Formal Description

We now present the channel abstraction that formalizes the desired service. The augmented secure channel **ASC** is described in Figure 3.2: The sender can provide a triple consisting of the explicit part of a header $E \in \mathcal{H}_{\mathrm{E}}$, the implicit part of the header $I \in \mathcal{H}_{\mathrm{I}}$, and a message $m \in \mathcal{M}$. The message remains confidential and the explicit part of the header is leaked at the adversarial interface. If the receiver knows the implicit part of the header, he can recover the message using the query $(\texttt{fetch}, I)$ and verify the authenticity of the message and both parts of the header. If the verification fails, the system stops delivering messages and signals an error by outputting $\perp$. The adversary has the ability to deliver messages and to inject a special element that will terminate the channel at the receiver's side once fetched. Delivering a message notifies the receiver of the new message and provides him with the explicit part of the header.

3.2.3 Construction

After motivating the need for the new channel **ASC**, we now show how to construct it using an AEAD-scheme from a shared-key resource $\mathbf{SK}_{\mathcal{K}}$ and an insecure channel **IC**. These assumed resources were introduced in Section 2.3.

Protocol

Recall that in constructive cryptography, a channel protocol is modeled as a pair of converters that specify the actions of both honest parties Alice and Bob. For an AEAD-scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, we present the protocol $(\mathsf{enc}_{\Pi}, \mathsf{dec}_{\Pi})$ as pseudocode in Figure 3.3. The converter for the sender, enc_{Π}, accepts inputs of the form $(\texttt{send}, E, I, m)$ at its outer interface and encrypts the message m using $\mathcal{E}$. The nonce is implemented as a

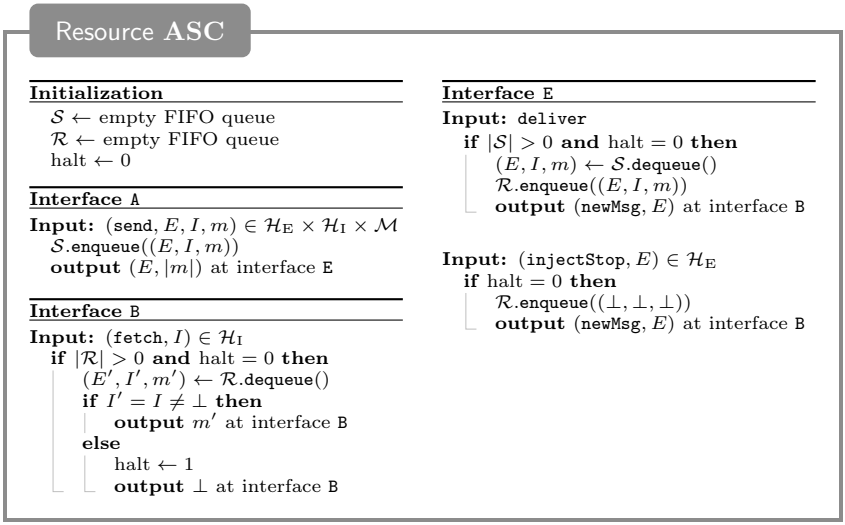


Figure 3.2: Description of **ASC**, an augmented secure channel.

counter, the additional data[2] is $A = (E, I)$ and the key is provided by the key-resource $\mathbf{SK}_\mathcal{K}$. An encoding of the resulting ciphertext and the explicit part of the header is output to the insecure channel **IC**.

The receiver converter dec_Π receives inputs from **IC** and queues the header-ciphertext pairs internally in a queue $\mathcal{Q}$. For each newly arrived message a notification is output at the outer interface. The next ciphertext c in the queue is decrypted if dec_Π is invoked with the implicit part of the corresponding header. The parameters for decryption are again the header as the additional data, the counter as the nonce and the shared key. On success, the corresponding plaintext is output at the outer interface. If decryption fails, the converter stops and signals an error by outputting $\perp$.

[2]Here, $(E, I) \in \mathcal{H}_\mathrm{E} \times \mathcal{H}_\mathrm{I}$ denotes an encoding of that pair as an element in $\mathcal{H}$. Abusing notation, we generally do not distinguish between a tuple and its encoding as an element in Σ^*.

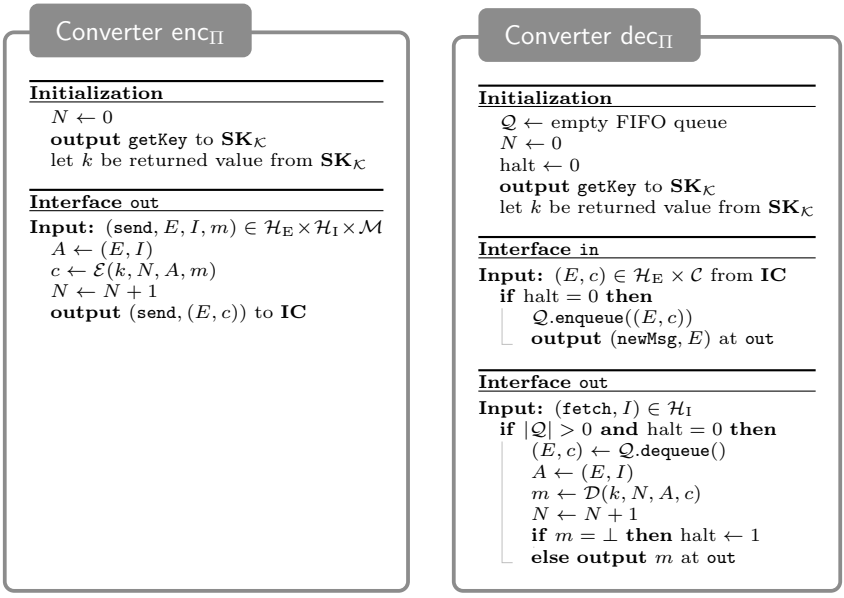


Figure 3.3: The protocol converters for the sender (left) and the receiver (right) that construct **ASC** via an AEAD-scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$.

Security Proof

The following two lemmata relate the AEAD-security game to the distinguishing advantage in the correctness and security condition, respectively. Recall Section 2.3.4 for the relevant conditions. Note that the correctness condition does not follow directly from the correctness of the AEAD-scheme. This is because it is not excluded that a ciphertext gets decrypted to some message $m \neq \perp$ if the wrong additional data is supplied, while the system dlv^E **ASC** always returns $\perp$ if the wrong value for I is input at interface B. We need the security of the AEAD-scheme to conclude that such invalid decryptions can only occur with small probability.

Lemma 3.2.1. *There is an (efficient) transformation ρ described in the proof that maps distinguishers* $\mathbf{D}$ *for two resources to valid adversaries* $\mathcal{A} =$

$\rho(\mathbf{D})$ *for the AEAD-security game such that*

$$\Delta^{\mathbf{D}}(\mathsf{enc}_\Pi{}^{\mathtt{A}}\mathsf{dec}_\Pi{}^{\mathtt{B}}\mathsf{dlv}^{\mathtt{E}}[\mathbf{SK}_\mathcal{K}, \mathbf{IC}], \mathsf{dlv}^{\mathtt{E}}\,\mathbf{ASC}) \leq \mathsf{Adv}^{\mathit{ae}}_\Pi(\rho(\mathbf{D})).$$

Proof. In $\mathsf{enc}_\Pi{}^{\mathtt{A}}\mathsf{dec}_\Pi{}^{\mathtt{B}}\mathsf{dlv}^{\mathtt{E}}[\mathbf{SK}_\mathcal{K}, \mathbf{IC}]$, the converter dlv is attached at interface $\mathtt{E}$ and answers any output produced by $\mathbf{IC}$ with the input $\mathtt{deliver}$. This essentially converts $\mathbf{IC}$ into a reliable transmission channel: whatever pair (E, c) is input by converter enc_Π, it is immediately delivered to dec_Π that outputs a notification $(\mathtt{newMsg}, E)$ at its outer interface. Furthermore, if the ith input at interface $\mathtt{A}$ is $(\mathtt{send}, E_i, I_i, m_i)$, and the ith input at interface $\mathtt{B}$ is $(\mathtt{fetch}, I_i)$, then the output at interface $\mathtt{B}$ is m_i. The same holds for system $\mathsf{dlv}^{\mathtt{E}}\mathbf{ASC}$. Only if the ith input at interface $\mathtt{B}$ is $(\mathtt{fetch}, I'_i)$ for $I'_i \neq I_i$, then the behavior of the two systems can differ: While $\mathsf{dlv}^{\mathtt{E}}\mathbf{ASC}$ always returns $\perp$ in this case, for $\mathsf{enc}_\Pi{}^{\mathtt{A}}\mathsf{dec}_\Pi{}^{\mathtt{B}}\mathsf{dlv}^{\mathtt{E}}[\mathbf{SK}_\mathcal{K}, \mathbf{IC}]$ it is possible that a message $m \neq \perp$ is returned. Since this is the only difference between the two systems, we can upper bound the distinguishing advantage by the probability that $\mathbf{D}$ can provoke such an output at interface $\mathtt{B}$ when interacting with $\mathsf{enc}_\Pi{}^{\mathtt{A}}\mathsf{dec}_\Pi{}^{\mathtt{B}}\mathsf{dlv}^{\mathtt{E}}[\mathbf{SK}_\mathcal{K}, \mathbf{IC}]$. It remains to bound the probability of this event, subsequently denoted by $\mathcal{F}$.

Note that $\mathcal{F}$ occurs exactly if the decryption algorithm of the AEAD-scheme returns a message $m \neq \perp$ on input a different additional data than used for encryption. Based on this observation, we build an adversary $\mathcal{A}$ that emulates a view towards distinguisher $\mathbf{D}$ that is identical to an interaction of $\mathbf{D}$ with $\mathsf{enc}_\Pi{}^{\mathtt{A}}\mathsf{dec}_\Pi{}^{\mathtt{B}}\mathsf{dlv}^{\mathtt{E}}[\mathbf{SK}_\mathcal{K}, \mathbf{IC}]$ if $\mathcal{A}$ gets access to its real oracles. The probability of provoking event $\mathcal{F}$ is preserved in this case. In contrast, if $\mathcal{A}$ gets access to the ideal oracles, the condition for event $\mathcal{F}$ cannot be satisfied as we argue below. This is a suitable distinguishing criterion.

More formally, the reduction ρ is defined as follows: The adversary $\mathcal{A} = \rho(\mathbf{D})$ initially sets $N_A, N_B \leftarrow 0$, initializes an empty FIFO queue $\mathcal{Q}$, and then emulates an execution to $\mathbf{D}$ as follows. When $\mathbf{D}$ inputs $(\mathtt{send}, E, I, m)$ at interface $\mathtt{A}$, $\mathcal{A}$ ask the query $(N_A, (I, E), m)$ to the oracle $\mathtt{Enc}$ to receive the answer c. It then executes $N_A \leftarrow N_A + 1$ and $\mathcal{Q}.\mathtt{enqueue}((E, I, m, c))$, and emulates the output $(\mathtt{newMsg}, E)$ at interface $\mathtt{B}$ for $\mathbf{D}$. Inputs $(\mathtt{fetch}, I')$ at interface $\mathtt{B}$ are ignored if $\mathcal{Q}$ is empty. Otherwise, $\mathcal{A}$ executes $(E, I, m, c) \leftarrow \mathcal{Q}.\mathtt{dequeue}()$. If $I' = I$, it sets $m' = m$; if $I' \neq I$, it asks the query $(N_B, (I', E), c)$ to the oracle $\mathtt{Dec}$ to receive the answer m'. It then sets $N_B \leftarrow N_B + 1$ and emulates the output m' at interface $\mathtt{B}$ for $\mathbf{D}$.

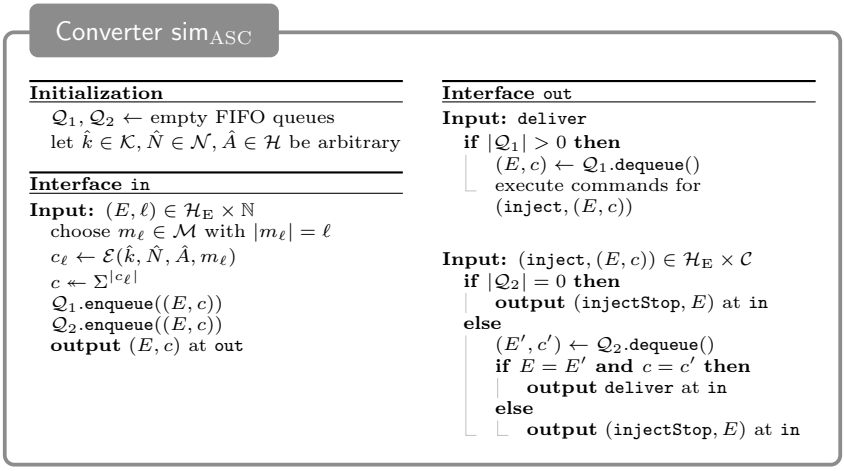


Figure 3.4: The simulator for the security condition of the construction of **ASC**.

If $m' \neq \bot$ and $I' \neq I$ (i.e., the event $\mathcal{F}$ occurs), $\mathcal{A}$ stops and returns 1. If $m' = \bot$, $\mathcal{A}$ ignores subsequent inputs at interface B. When $\mathbf{D}$ outputs a bit and $\mathcal{F}$ has not occurred, $\mathcal{A}$ returns 0. Observe that if $\mathcal{A}$ gets access to the ideal oracles, the conditions of event $\mathcal{F}$ cannot be met. We conclude the proof by noting that $\mathcal{A}$ is a valid adversary and $\mathsf{Adv}^{\mathsf{ae}}_{\Pi}(\rho(\mathbf{D}))$ equals the probability of the event $\mathcal{F}$. □

The next lemma implies the security condition of the construction:

Lemma 3.2.2. *For the simulator* $\mathsf{sim}_{\mathrm{ASC}}$ *defined in Figure 3.4, there is an (efficient) transformation* ρ' *described in the proof that maps distinguishers* $\mathbf{D}$ *for two resources to valid adversaries* $\mathcal{A} = \rho'(\mathbf{D})$ *for the AEAD-security game such that*

$$\Delta^{\mathbf{D}}(\mathsf{enc}_{\Pi}{}^{\mathtt{A}}\mathsf{dec}_{\Pi}{}^{\mathtt{B}}\,[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}], \mathsf{sim}^{\mathtt{E}}_{\mathrm{ASC}}\,\mathbf{ASC}) \leq \mathsf{Adv}^{\mathsf{ae}}_{\Pi}(\rho'(\mathbf{D})).$$

Proof. Let $\mathbf{D}$ be a distinguisher for $\mathsf{enc}_{\Pi}{}^{\mathtt{A}}\mathsf{dec}_{\Pi}{}^{\mathtt{B}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]$ and $\mathsf{sim}^{\mathtt{E}}_{\mathrm{ASC}}\mathbf{ASC}$. We define an adversary $\mathcal{A} = \rho'(\mathbf{D})$ for the AEAD-security game as follows. The adversary $\mathcal{A}$ initially sets $N_A, N_B, \mathtt{flag} \leftarrow 0$, initializes an empty

FIFO queue $\mathcal{S}$ and two empty lists[3] $\mathcal{L}$ and $\mathcal{R}$, and then emulates an execution of $\mathbf{D}$ by translating inputs of the distinguisher to oracle queries as well as answers from the oracles to outputs of the resource for $\mathbf{D}$. There are four types of inputs $\mathbf{D}$ can make:

$(\texttt{send}, E, I, m)$ **at interface** `A`**:** If $\mathcal{R}$ contains strictly less than $N_A + 1$ elements, $\mathcal{A}$ asks the query $(N_A, (E, I), m)$ to the oracle `Enc` and receives the answer c. It then stores (N_A, E, I, m, c) in the list $\mathcal{L}$, emulates the output (E, c) at interface `E` for $\mathbf{D}$, sets $N_A \leftarrow N_A + 1$, and executes $\mathcal{S}.\texttt{enqueue}((E, c))$.

If $\mathcal{R}$ contains at least N_A+1 elements, there is a pair $\mathcal{R}[N_A] = (E, c)$. $\mathcal{A}$ asks the query $(N_A, (E, I), c)$ to the oracle `Dec` to receive the plaintext m. If $m \neq \perp$, $\mathcal{A}$ sets $\texttt{flag} \leftarrow 1$, returns 1 as its decision and halts. If $m = \perp$, the tuple $(N_A, E, I, \perp, c)$ is stored in $\mathcal{L}$ and $\mathcal{A}$ asks the query $(N_A, (E, I), m)$ to the oracle `Enc`, receives the answer c and stores (N_A, E, I, m, c) in the list $\mathcal{L}$. Finally, $\mathcal{A}$ emulates the output (E, c) at interface `E` for $\mathbf{D}$, sets $N_A \leftarrow N_A + 1$, and executes $\mathcal{S}.\texttt{enqueue}((E, c))$.

`deliver` **at interface** `E`**:** If $|\mathcal{S}| > 0$, $\mathcal{A}$ executes $(E, c) \leftarrow \mathcal{S}.\texttt{dequeue}()$ followed by $\mathcal{R} \leftarrow \mathcal{R} \parallel (E, c)$. If $\perp$ has not been output at interface `B`, $\mathcal{A}$ emulates the output $(\texttt{newMsg}, E)$ at interface `B`.

$(\texttt{inject}, (E, c))$ **at interface** `E`**:** The adversary $\mathcal{A}$ executes $\mathcal{R} \leftarrow \mathcal{R} \parallel (E, c)$. If $\perp$ has not been output at interface `B`, $\mathcal{A}$ emulates the output $(\texttt{newMsg}, E)$ at interface `B`.

$(\texttt{fetch}, I)$ **at interface** `B`**:** If $\mathcal{R}$ is empty, the input is ignored. Otherwise, $\mathcal{A}$ executes $(E, c) \leftarrow \mathcal{R}[N_B]$. If $(N_B, E, I, \perp, c)$ is in $\mathcal{L}$, $\mathcal{A}$ emulates the output $\perp$ at interface `B` and ignores subsequent inputs at interface `B`. If (N_B, E, I, m, c) is in $\mathcal{L}$ for some $m \in \mathcal{M}$, $\mathcal{A}$ emulates the output m at interface `B` for $\mathbf{D}$ and sets $N_B \leftarrow N_B + 1$. Otherwise, $\mathcal{A}$ asks the query $(N_B, (E, I), c)$ to the oracle `Dec` to receive the plaintext m. The output m is emulated at interface `B` and the counter N_B is incremented. If $m = \perp$, $\mathcal{A}$ ignores subsequent inputs at interface `B`.

[3]For a list L, we denote by $L \parallel x$ the list L with x appended. Furthermore, the ith element of a list L with n elements is denoted by $L[i]$ for $i \in \{0, \ldots, n-1\}$.

When $\mathbf{D}$ outputs a bit b and if $\texttt{flag} = 0$, $\mathcal{A}$ returns b and halts. Note that $\mathcal{A}$ is a valid adversary since it asks at most one `Enc` and `Dec` query for each nonce (and therefore does not repeat queries) and never asks a query to the oracle `Dec` for a ciphertext that has been returned by a query to `Enc` for the same parameters (because for such ciphertext, the corresponding tuple is in the list $\mathcal{L}$). To analyze the success probability of $\mathcal{A}$, let F be the random variable that takes on the value of `flag` at the end of the random experiment between $\mathcal{A}$ and $\mathbf{Real}^{\mathrm{ae}}_{\mathbf{\Pi}}$.

We claim that the view of $\mathbf{D}$ when connected to $\mathsf{sim}^{\mathtt{E}}_{\mathrm{ASC}}\mathbf{ASC}$ is identical to the view emulated by $\mathcal{A}$ with access to the ideal oracles. Additionally, the view of $\mathbf{D}$ when connected to $\mathsf{enc}_{\Pi}{}^{\mathtt{A}}\mathsf{dec}_{\Pi}{}^{\mathtt{B}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]$ is identical to the view emulated by $\mathcal{A}$ with access to the real oracles as long as $\texttt{flag} = 0$. This implies the statement of the lemma:

$$
\begin{aligned}
&\mathsf{Adv}^{\mathsf{ae}}_{\Pi}(\mathcal{A}) = \Pr\left[\mathcal{A}^{\mathbf{Real}^{\mathrm{ae}}_{\mathbf{\Pi}}} = 1\right] - \Pr\left[\mathcal{A}^{\mathbf{Ideal}^{\mathrm{ae}}_{\mathbf{\Pi}}} = 1\right] \\
&= \Pr[F = 1] + \Pr[F = 0] \cdot \underbrace{\Pr\left[\mathcal{A}^{\mathbf{Real}^{\mathrm{ae}}_{\mathbf{\Pi}}} = 1 \mid F = 0\right]}_{= \Pr[\mathbf{D}(\mathsf{enc}_{\Pi}{}^{\mathtt{A}}\mathsf{dec}_{\Pi}{}^{\mathtt{B}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]) = 1]} - \Pr\left[\mathcal{A}^{\mathbf{Ideal}^{\mathrm{ae}}_{\mathbf{\Pi}}} = 1\right] \\
&\geq \Pr\left[\mathbf{D}\left(\mathsf{enc}_{\Pi}{}^{\mathtt{A}}\mathsf{dec}_{\Pi}{}^{\mathtt{B}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]\right) = 1\right] - \Pr\left[\mathbf{D}\left(\mathsf{sim}^{\mathtt{E}}_{\mathrm{ASC}}\,\mathbf{ASC}\right) = 1\right] \\
&= \Delta^{\mathbf{D}}(\mathsf{enc}_{\Pi}{}^{\mathtt{A}}\mathsf{dec}_{\Pi}{}^{\mathtt{B}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}], \mathsf{sim}^{\mathtt{E}}_{\mathrm{ASC}}\,\mathbf{ASC}).
\end{aligned}
$$

To prove this claim, we distinguish the possible inputs by $\mathbf{D}$ and compare the resulting outputs:

$(\texttt{send}, E, I, m)$ **at interface `A`:** In system $\mathsf{enc}_{\Pi}{}^{\mathtt{A}}\mathsf{dec}_{\Pi}{}^{\mathtt{B}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]$, the converter enc_{Π} evaluates $c \leftarrow \mathcal{E}(k, N, (E, I), m)$, where N is the number of sent messages before this input. The explicit part E of the header is sent together with c over $\mathbf{IC}$, which outputs the pair (E, c) at interface `E`. The same output is emulated by $\mathcal{A}$ in the real game since the oracle `Enc` in this case also evaluates the algorithm $\mathcal{E}$.

In the system $\mathsf{sim}^{\mathtt{E}}_{\mathrm{ASC}}\,\mathbf{ASC}$, the triple (E, I, m) is inserted into the senders queue of $\mathbf{ASC}$ and the pair $(E, |m|)$ is output to the simulator $\mathsf{sim}_{\mathrm{ASC}}$, which in turn generates a uniformly random ciphertext c of the same length as ciphertexts for m. Note that by Definition 3.1.1, the length of ciphertexts only depend on the length of the message, so the values of $\hat{k}$, $\hat{N}$, and $\hat{A}$ used by $\mathsf{sim}_{\mathrm{ASC}}$ to determine this length are irrelevant. The simulator then stores (E, c) in its own queue for later reference and outputs this pair at

interface E. Note that Enc in $\mathbf{Ideal}^{\mathrm{ae}}_{\Pi}$ generates ciphertexts with the same distribution, so the view emulated by $\mathcal{A}$ is identical.

deliver at interface E: If the sender's queue it non-empty, the next element (E, c) is dequeued from it and $\mathbf{D}$ receives the output $(\mathtt{newMsg}, E)$ from interface B if there has not been an output $\perp$ in both systems and in the emulated view.

$(\mathtt{inject}, (E, c))$ at interface E: In $\mathsf{enc}_{\Pi}{}^{\mathtt{A}}\mathsf{dec}_{\Pi}{}^{\mathtt{B}}\,[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]$, the injected pair is inserted into the receiver's queue of the converter dec_{Π} and the notification $(\mathtt{newMsg}, E)$ is output at interface B.

In $\mathsf{sim}^{\mathtt{E}}_{\mathrm{ASC}}\,\mathbf{ASC}$, the simulator checks whether the injected pair is equal to the top-element (E', c') of its queue $\mathcal{Q}_2$. If this is the case, the simulator outputs deliver with the effect that the notification $(\mathtt{newMsg}, E)$ is output at interface B. If $(E, c) \neq (E', c')$, $\mathsf{sim}_{\mathrm{ASC}}$ injects a stop element by $(\mathtt{injectStop}, E)$, which also yields the output $(\mathtt{newMsg}, E)$ at interface B. Note that by definition of **ASC**, this element is guaranteed to yield $\perp$ when fetched at interface B.

We see that in the emulation by $\mathcal{A}$, $\mathbf{D}$ receives $(\mathtt{newMsg}, E)$ from interface B if there has not been an output $\perp$ before at interface B. The same holds for both systems $\mathsf{enc}_{\Pi}{}^{\mathtt{A}}\mathsf{dec}_{\Pi}{}^{\mathtt{B}}\,[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]$ and $\mathsf{sim}^{\mathtt{E}}_{\mathrm{ASC}}\,\mathbf{ASC}$ in an interaction with $\mathbf{D}$.

$(\mathtt{fetch}, I)$ at interface B: Assume this is the ith input at interface B, there have been at least i inputs deliver or inject at interface E, and there has not been an output $\perp$ so far (otherwise the input is always ignored). In the system $\mathsf{enc}_{\Pi}{}^{\mathtt{A}}\mathsf{dec}_{\Pi}{}^{\mathtt{B}}\,[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]$, the converter dec_{Π} retrieves the top element of its queue. This value is equal to the ith delivered or injected pair (E, c) at interface E. The converter dec_{Π} then computes $m \leftarrow \mathcal{D}(k, i-1, (E, I), c)$ and outputs m.

In the view emulated by $\mathcal{A}$ in the real game, (E, c) also corresponds to the ith delivered or injected pair. If the tuple $(i-1, E, I, m, c)$ is found in $\mathcal{L}$ for some $m \in \mathcal{M}$, m is output at interface B. By the correctness of the AEAD-scheme, m is then equal to the output of the algorithm $\mathcal{D}$ for the corresponding parameters. If no such tuple is in $\mathcal{L}$, $\mathcal{A}$ decrypts c with the corresponding parameters using the

oracle Dec and also outputs the resulting message at interface B. Since the real oracle Dec evaluates $\mathcal{D}$, we conclude that the views are identical in this case.

In $\mathsf{sim}^{\mathtt{E}}_{\mathrm{ASC}}$ **ASC**, the resource checks whether $I = I'$, where (E', I', m') is the next element in the queue $\mathcal{R}$. If this is the case, it outputs m' at interface B, otherwise it outputs $\perp$. Furthermore, only those elements can be successfully fetched that do not correspond to stop-elements $(\perp, \perp, \perp)$. By construction of the simulator, the ith element of the sender's queue is only delivered if the ith injected pair (E, c) at interface E matches the simulated pair output at interface E in reaction to the ith input $(\mathtt{send}, E', I', m')$ at interface A. In any other case, a stop-element is injected into the receiver's queue.

To determine whether the values of the ith injection match the simulated values for the ith input at interface A, $\mathsf{sim}_{\mathrm{ASC}}$ maintains the queue $\mathcal{Q}_2$ such that its top element, after i injections, stores exactly these values. Note that the queue $\mathcal{Q}_1$ on the other hand is only needed to simulate the queue of the insecure channel **IC** in the real world and to figure out the next message in the simulation of a deliver-request.

In the view emulated by $\mathcal{A}$ in the ideal game, the list $\mathcal{L}$ ensures that the same message m' is output at interface B if all the values match as above. Furthermore, if there is not a match, the output is $\perp$ because the ideal oracle Dec always returns $\perp$. In particular, the condition that the ith simulated pair correspond to the ith injected pair is equivalent to requiring that the tuple $(i-1, E, I, m, c)$, for some message m, is an element of $\mathcal{L}$.

Hence, the views for **D** are also identical in this case.

This concludes the proof of the claim and thus of the lemma. □

The results from Lemma 3.2.1 and Lemma 3.2.2 can be summarized as follows:

Theorem 3.2.3. *The protocol* $(\mathsf{enc}_\Pi, \mathsf{dec}_\Pi)$ *constructs* **ASC** *from* $[\mathbf{SK}_\mathcal{K}, \mathbf{IC}]$. *More specifically, we have for the simulator* $\mathsf{sim}_{\mathrm{ASC}}$ *in Figure 3.4 and for*

all distinguishers $\mathbf{D}$

$$\Delta^{\mathbf{D}}(\mathsf{enc}_{\Pi}{}^{\mathtt{A}}\mathsf{dec}_{\Pi}{}^{\mathtt{B}}\mathsf{dlv}^{\mathtt{E}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}], \mathsf{dlv}^{\mathtt{E}}\,\mathbf{ASC}) \leq \mathsf{Adv}^{\mathsf{ae}}_{\Pi}(\rho(\mathbf{D}))$$

and $$\Delta^{\mathbf{D}}(\mathsf{enc}_{\Pi}{}^{\mathtt{A}}\mathsf{dec}_{\Pi}{}^{\mathtt{B}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}], \mathsf{sim}^{\mathtt{E}}_{\mathrm{ASC}}\,\mathbf{ASC}) \leq \mathsf{Adv}^{\mathsf{ae}}_{\Pi}(\rho'(\mathbf{D})),$$

where ρ *and* ρ' *are the reductions defined in the proofs of Lemma 3.2.1 and Lemma 3.2.2, respectively.*

3.2.4 Application: On TLS Security

A long line of work analyzes the security of TLS (mainly versions prior to 1.3) [GMP+08, HSD+05, JKSS12, KMO+15, KPW13, MSW08, PRS11, WS96]. Several recent papers [JKSS12, KPW13] use a security notion called Authenticated and Confidential Channel Establishment (ACCE), a game-based definition that models both the handshake and the record layer at once. Motivated by the adoption of AEAD as the core of the record-layer sub-protocol in TLS 1.3, we give a novel interpretation for the goal of (all drafts of) the TLS record layer: constructing a specific instantiation of an ASC, from insecure communication and a shared secret key. In fact, depending on the context, messages in the TLS record protocol may consist of private and non-private parts, which are both authenticated.

We show how a generic ASC construction directly leads to this specific instantiation of an ASC. We thereby obtain a provably secure TLS record protocol. As we will see, our analysis also covers slight variations of the core protocol itself. Indeed, [BMM+15a] proposed small improvements over the then-draft of TLS 1.3 (August 2015) and some of those improvements are still found in the most recent draft.

The Structure of the TLS 1.3 Record Layer

Version 1.3 of TLS is about to be standardized at the time of writing these lines. The standardization process has taken several years and has gone through a sequence of developments. In comparison to TLS 1.2, the new version of the record payload protection protocol mandates AEAD ciphers[4] and the format of the authenticated data has changed.

[4]Previous versions of TLS supported MAC-then-Encrypt modes.

The core question during the development process has always been: What does it mean for the TLS record layer to be secure? We propose a simple answer to this question in the form of a new channel abstraction.

Each draft of the TLS 1.3 record layer specifies a basic structure for payload packets: each such packet contains a payload and has a type flag. While in early drafts, this type flag characterized some of the context of the payload (e.g., alert messages or application messages), it is now merely a placeholder (i.e., fixed to the number 23) to harden traffic flow analysis.[5] Our analysis leaves the exact role of the type flag open to make a more general statement and to allow for a unified proof for a range of choices of how to use the type flag.

In any case, while the entire packet is authenticated, only the content of the packet has to be private and hidden from the attacker. This resembles a specific type of channel: a secure channel where messages are tagged with a non-private type-flag from the set of types $\mathcal{T} := \{0, \ldots, 255\}$. The TLS record payload protection can be considered secure if it provably constructs this secure channel. We formalize this channel as the resource $\mathbf{SEC}_{\mathrm{TLS}}$ and provide a formal description thereof in Figure 3.5.[6]

Note that in contrast to **ASC**, the channel $\mathbf{SEC}_{\mathrm{TLS}}$ does not contain an implicit part of the header and messages are directly delivered to Bob without the need to fetch them. Therefore, $\mathbf{SEC}_{\mathrm{TLS}}$ does not allow the authentication of data without sending it. One can thus view $\mathbf{SEC}_{\mathrm{TLS}}$ as an augmented secure channel that is more restricted than **ASC** but also simpler to use.

The Record-Layer Construction

In this section, we present a construction of the channel $\mathbf{SEC}_{\mathrm{TLS}}$ from **ASC**. To this end, we introduce the protocol $(\mathsf{tlsSnd}, \mathsf{tlsRcv})$, which is described in Figure 3.6 and manages the usage of the resource **ASC**. The

[5] We refer to the draft that is available for download at `https://tools.ietf.org/html/draft-ietf-tls-tls13-28`.

[6] While applications usually provide data to TLS as a sequence of multi-byte strings, TLS only guarantees that the same stream of bytes, as the concatenation of the individual strings, is delivered. TLS does not guarantee that the boundaries between the multi-byte strings are preserved as chosen by the application, cf. [FGMP15]. The message m in Figure 3.5 is to be understood as the multi-byte string used within the TLS protocol, which is not necessarily the same as chosen by the higher-level application.

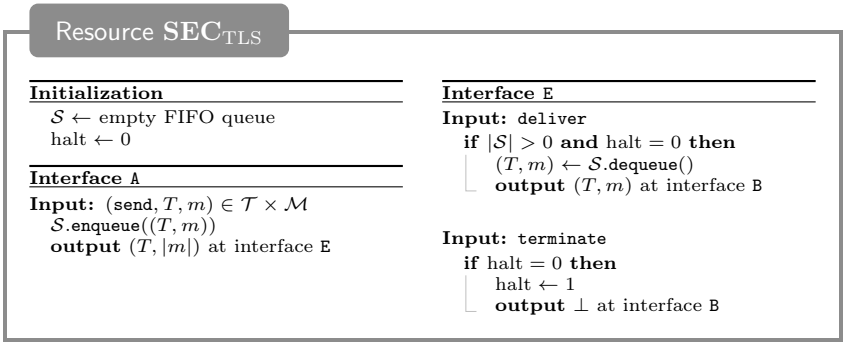


Figure 3.5: Description of the channel $\mathbf{SEC}_{\mathrm{TLS}}$.

implicit part I of the header contains the protocol version field V of the TLS record layer (in the current draft it is a legacy value consisting of two bytes which we generically denote $v.\mathsf{X}$) and the explicit part E consists of the type flag T (one byte) which the caller can define for the sake of generality of the statement. We denote the (generic) type space by $\mathcal{T}$.

Theorem 3.2.4. *The protocol* $(\mathsf{tlsSnd}, \mathsf{tlsRcv})$ *constructs* $\mathbf{SEC}_{\mathrm{TLS}}$ *from* **ASC**. *More specifically, we have for the simulator* $\mathsf{sim}_{\mathrm{TLS}}$ *defined in Figure 3.7 and for all distinguishers* $\mathbf{D}$

$$\Delta^{\mathbf{D}}(\mathsf{tlsSnd}^{\mathtt{A}}\mathsf{tlsRcv}^{\mathtt{B}}\mathsf{dlv}^{\mathtt{E}}\mathbf{ASC}, \mathsf{dlv}^{\mathtt{E}}\mathbf{SEC}_{\mathrm{TLS}}) = 0 \tag{3.1}$$

and

$$\Delta^{\mathbf{D}}(\mathsf{tlsSnd}^{\mathtt{A}}\mathsf{tlsRcv}^{\mathtt{B}}\mathbf{ASC}, \mathsf{sim}^{\mathtt{E}}_{\mathrm{TLS}}\mathbf{SEC}_{\mathrm{TLS}}) = 0. \tag{3.2}$$

Proof. The correctness condition (3.1) is easy to verify: On input $(\mathtt{send}, T, m)$ at interface A, the system $\mathsf{dlv}^{\mathtt{E}}\mathbf{SEC}_{\mathrm{TLS}}$ directly outputs (T, m) at interface B. The same holds for system $\mathsf{tlsSnd}^{\mathtt{A}}\mathsf{tlsRcv}^{\mathtt{B}}\mathsf{dlv}^{\mathtt{E}}\mathbf{ASC}$: On input $(\mathtt{send}, T, m)$, the converter tlsSnd inputs $(\mathtt{send}, T, V, m)$ to **ASC**. The converter tlsRcv then obtains the notification $(\mathtt{newMsg}, T)$ and queries $(\mathtt{fetch}, V)$ to **ASC**, which results in the output m from **ASC**, which in turn triggers tlsRcv to output (T, m). Since the two systems behave identically, every distinguisher has advantage 0 in distinguishing them, i.e., (3.1) follows.

To verify the security condition (3.2), we distinguish the possible inputs to the system:

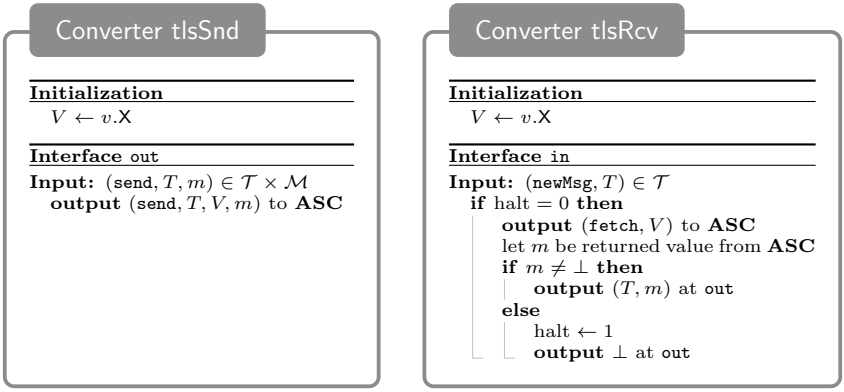


Figure 3.6: The protocol converters for the sender (left) and the receiver (right) that construct $\mathbf{SEC}_{\mathrm{TLS}}$ from **ASC**.

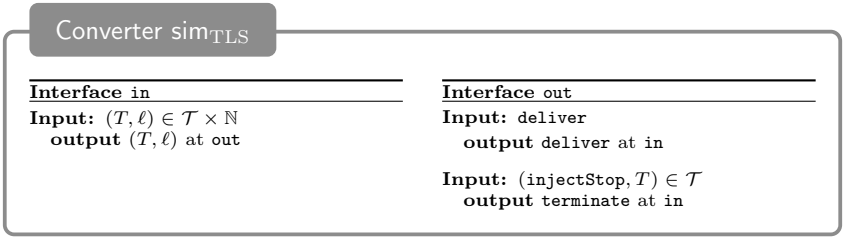


Figure 3.7: The simulator for the security condition of the construction of $\mathbf{SEC}_{\mathrm{TLS}}$.

Input $(\texttt{send}, T, m)$ **at interface** `A`**:** In the system $\mathsf{tlsSnd}^{\mathtt{A}}\mathsf{tlsRcv}^{\mathtt{B}}\mathbf{ASC}$, this input results in the converter tlsSnd inputting $(\texttt{send}, T, V, m)$ to **ASC**, which yields the output $(T, |m|)$ at interface `E` of **ASC**. In $\mathsf{sim}^{\mathtt{E}}_{\mathrm{TLS}}\mathbf{SEC}_{\mathrm{TLS}}$, the values $(T, |m|)$ are given to the simulator, which then outputs $(T, |m|)$ at its outer interface.

Input `deliver` **at interface** `E`**:** In $\mathsf{tlsSnd}^{\mathtt{A}}\mathsf{tlsRcv}^{\mathtt{B}}\mathbf{ASC}$, if the queue $\mathcal{S}$ in **ASC** is empty, nothing happens. Otherwise, the converter tlsRcv receives the notification $(\texttt{newMsg}, T)$. Then, tlsRcv inputs $(\texttt{fetch}, V)$ to **ASC** if it has not already halted. In this case, there have been only inputs `deliver` at interface `E` and therefore the verification within **ASC** succeeds. Thus, tlsRcv obtains the message m and outputs (T, m).

In $\mathsf{sim}^{\mathtt{E}}_{\mathrm{TLS}}\mathbf{SEC}_{\mathrm{TLS}}$, the simulator inputs `deliver` to $\mathbf{SEC}_{\mathrm{TLS}}$. If $\mathcal{S}$ in $\mathbf{SEC}_{\mathrm{TLS}}$ is empty, nothing happens. Otherwise, the next tuple (T, m) in $\mathcal{S}$ is output at interface `B` if the channel has not halted before.

Input $(\texttt{injectStop}, T)$ **at interface** `E`**:** In the system $\mathsf{tlsSnd}^{\mathtt{A}}\mathsf{tlsRcv}^{\mathtt{B}}\mathbf{ASC}$, the notification $(\texttt{newMsg}, T)$ is output to tlsRcv. The converter tlsRcv then outputs $(\texttt{fetch}, V)$ to **ASC** and since the element is an inserted empty element, the verification within **ASC** fails and tlsRcv outputs $\perp$ and stops by setting halt $\leftarrow 1$.

In $\mathsf{sim}^{\mathtt{E}}_{\mathrm{TLS}}\mathbf{SEC}_{\mathrm{TLS}}$, $\mathsf{sim}_{\mathrm{TLS}}$ terminates the session, which causes the output $\perp$ at interface `B` and results in no further messages being processed by Bob.

To see that the two described systems behave identically, we only have to observe that they both terminate the session if an empty message is injected into the channel and that all inputs are delivered in order until termination. We again conclude that every distinguisher has advantage 0 in distinguishing these systems, i.e., we obtain (3.2). This completes the proof. □

Putting Everything Together

We have shown that $(\mathsf{tlsSnd}, \mathsf{tlsRcv})$ constructs $\mathbf{SEC}_{\mathrm{TLS}}$ from **ASC**. Since by Theorem 3.2.3, the protocol $(\mathsf{enc}_{\Pi}, \mathsf{dec}_{\Pi})$ constructs the channel **ASC**

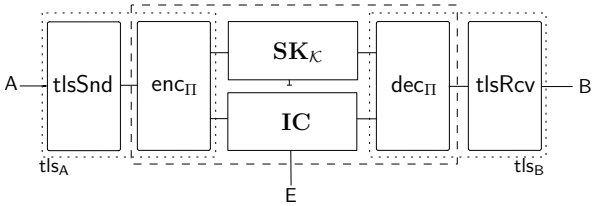


Figure 3.8: Illustration of the composed protocol $(\mathsf{tls}_\mathsf{A}, \mathsf{tls}_\mathsf{B})$. For a secure AEAD-scheme, the resource $\mathsf{enc}_\Pi{}^\mathsf{A}\mathsf{dec}_\Pi{}^\mathsf{B}[\mathbf{SK}_\mathcal{K}, \mathbf{IC}]$ inside the dashed box in the center is indistinguishable from $\mathsf{sim}^\mathsf{E}_{\mathrm{ASC}}\mathbf{ASC}$.

from a shared secret key and an insecure channel, we can invoke the composition theorem of constructive cryptography to conclude that the composition of both protocols constructs $\mathbf{SEC}_{\mathrm{TLS}}$ from a shared key $\mathbf{SK}_\mathcal{K}$ and an insecure channel $\mathbf{IC}$. Note that the key resource $\mathbf{SK}_\mathcal{K}$ is constructed by the handshake protocol if both parties are authenticated [KMO+15]. See Figure 3.8 for a graphical illustration of the composed protocol $(\mathsf{tls}_\mathsf{A}, \mathsf{tls}_\mathsf{B})$.

The protocol for the sender tls_A works as follows: On input $(\mathtt{send}, T, m)$, the message m is encrypted with a call to the AEAD-scheme as $c \leftarrow \mathcal{E}(k, N, A, m)$, where k is the shared key retrieved from $\mathbf{SK}_\mathcal{K}$, N is the internal counter and $A = (T, V)$ is the additional data. Finally, the pair (T, c) is sent over the insecure channel.

The protocol for the receiver tls_B works analogously: On input a new pair (T, c) from $\mathbf{IC}$, the ciphertext is decrypted to $m \leftarrow \mathcal{D}(k, N, A, c)$, where N is the internal counter, A is the additional data, and k is the shared key as above. Note that the implicit part of the header is fixed and provided by tlsRcv immediately after receiving the notification $(\mathtt{newMsg}, T)$ from dec_Π.

In summary, the protocol $(\mathsf{tls}_\mathsf{A}, \mathsf{tls}_\mathsf{B})$ provably achieves the goal of the TLS record layer. This reveals some insights in how to get the details right in defining the record layer. We mention three examples:

(A): The nonce of the AEAD scheme can be a simple counter value left-padded with zeros to be of the appropriate length;

(B): The sequence number does not have to be included in the additional data part;

(C): The version number does not need to be transmitted explicitly as

part of the TLS record (after the handshake). It can be included as part of the additional data for clarity and this would not harm traffic-flow confidentiality since it is part of the implicit header.

While earlier drafts did not incorporate the above three properties, the current draft explicitly supports property (A), where the simple counter might start at an arbitrary offset, and property (B). However, it leaves a version-number field explicit in the header of transmitted packets for legacy reasons. Note that the proof for the case where the version number field is moved to the explicit part of the header is essentially identical to the one presented in this section.

Chapter 4

Robust Authenticated Encryption

4.1 Introduction

We continue our study on practical cryptographic building blocks with a closer look at robust authenticated encryption. We introduce an equivalent, constructive security definition that allows us to study what exactly this primitive achieves. Furthermore, we provide an idealization that provably captures the best-possible guarantees that are achievable by any symmetric scheme to protect communication.

4.1.1 Motivation

Since its introduction, several notions of authenticated encryption have emerged in a series of works [BN00, BR00, GD02, Jut01, KY01, RBBK01], including authenticated encryption with associated data [BRW04, Rog02], which is the topic of Chapter 3, and more advanced notions such as misuse-resistant authenticated encryption [RS06]. In this development, *robust authenticated encryption (RAE)*, introduced by Hoang, Krovetz, and Rogaway [HKR15], is the latest and most ambitious notion. Robust authenticated encryption allows to specify the ciphertext expansion λ that determines how much longer ciphertexts are compared to the corre-

sponding plaintexts. Its self-declared goal in [HKR15] is to provide the best-possible authenticity and confidentiality for every choice of λ. This raises the question of what best-possible authenticity and confidentiality is, and whether RAE actually achieves it. We provide a formal model that allows us to investigate this question and answer it in the affirmative. We further show how to use verifiable redundancy to improve security, and we show what security guarantees remain if values intended as nonces are reused. Both questions were addressed in [HKR15] but not proven formally.

Robust Authenticated Encryption. An RAE scheme consists of a key distribution $\mathcal{K}$, a deterministic encryption algorithm $\mathcal{E}$, and a deterministic decryption algorithm $\mathcal{D}$. The encryption algorithm takes as input a key k, a nonce N, associated data A, the ciphertext expansion λ, and a message m. It outputs a ciphertext c. The decryption algorithm takes as input k, N, A, λ, and c, and returns the corresponding message m (or $\perp$ if c is an invalid ciphertext). In [HKR15], the security of an RAE scheme is defined via a game in which an adversary has access to two oracles and has to distinguish between two possible settings. In the first setting, the oracles correspond to the encryption and decryption algorithm of the RAE scheme, where the key is fixed in the beginning and chosen according to $\mathcal{K}$. In the second setting, the first oracle chooses for each N, A, λ, and message length ℓ an injective function that maps strings of length ℓ to strings of length $\ell + \lambda$. On input (N, A, λ, m), the oracle answers by evaluating the corresponding function. The second oracle corresponds to the partially defined inverse of that function that answers $\perp$ if the given value has no preimage. An RAE scheme is secure if these two settings are indistinguishable for efficient adversaries. We note that while this seems to be a strong guarantee, it is not clear which security such a scheme actually provides in a specific application and whether it is best-possible.

4.1.2 RAE: Standard Definition

We define the syntax of a robust authenticated encryption scheme following [HKR15]. As in previous sections, Σ denotes an alphabet and for a string $x \in \Sigma^*$, $|x|$ denotes its length.

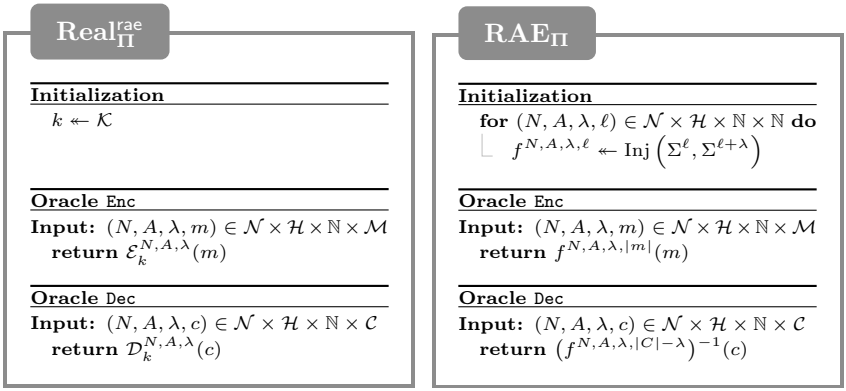


Figure 4.1: Security games for the RAE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ as defined in [HKR15].

Definition 4.1.1. A *robust authenticated encryption (RAE) scheme* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of a key distribution $\mathcal{K}$, a deterministic encryption algorithm $\mathcal{E}$ that maps a key $k \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, associated data $A \in \mathcal{H}$, ciphertext expansion $\lambda \in \mathbb{N}$, and a message $m \in \mathcal{M}$ to a ciphertext $c \in \mathcal{C}$, and a deterministic decryption algorithm $\mathcal{D}$ that maps (K, N, A, λ, c) to an element in $\mathcal{M} \cup \{\perp\}$. We assume the domains $\mathcal{N}$, $\mathcal{H}$, $\mathcal{M}$, and $\mathcal{C}$ are equal to Σ^*. We write $\mathcal{E}_k^{N,A,\lambda}$ and $\mathcal{D}_k^{N,A,\lambda}$ for the functions $\mathcal{E}(k, N, A, \lambda, \cdot)$ and $\mathcal{D}(k, N, A, \lambda, \cdot)$, respectively. We require that $\mathcal{D}_k^{N,A,\lambda}\left(\mathcal{E}_k^{N,A,\lambda}(m)\right) = m$ for all k, N, A, λ, m.

The security of an RAE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is defined in [HKR15] via the reference games $\mathbf{Real}_{\mathbf{\Pi}}^{\mathrm{rae}}$ and $\mathbf{RAE}_{\mathbf{\Pi}}$ depicted in Figure 4.1. The game $\mathbf{Real}_{\mathbf{\Pi}}^{\mathrm{rae}}$ provides oracle access to $\mathcal{E}$ and $\mathcal{D}$, and $\mathbf{RAE}_{\mathbf{\Pi}}$ provides oracle access to ideal uniform random injections and their inverses.

Definition 4.1.2. The game-based RAE-advantage of an adversary $\mathcal{A}$ is defined as

$$\mathsf{Adv}_{\Pi}^{\mathsf{rae\text{-}game}}(\mathcal{A}) := \Pr\left[\mathcal{A}^{\mathbf{Real}_{\mathbf{\Pi}}^{\mathrm{rae}}} = 1\right] - \Pr\left[\mathcal{A}^{\mathbf{RAE}_{\mathbf{\Pi}}} = 1\right].$$

4.1.3 The Constructive Cryptography Setting

Like in Chapter 3, we consider the Alice-Bob-Eve setting of constructive cryptography as introduced in Sections 2.3.4 and 2.3.5.That is, resources in this chapter have the interface set $\mathcal{I} = \mathcal{P} \cup \{\mathtt{E}\}$, where the set of interfaces with honest behavior is $\mathcal{P} = \{\mathtt{A}, \mathtt{B}\}$.

4.1.4 Specific Contributions

In the vein of Chapter 3 and accounting for the associated data RAE schemes support, the obvious goal of RAE can be defined as constructing a (possibly special type of) augmented secure channel (ASC) from a shared secret key and an insecure channel. Recall that an ASC takes as input from the sender a tuple (A, m), leaks A and the length of m to the adversary, and allows the adversary to either deliver the pair (A, m) or to terminate the channel. This channel provides authenticity for both A and m, but confidentiality is only guaranteed for the message m. The value A can for example be used to authenticate non-private header information.

Uniform random injection resource. Instead of directly constructing channels from a shared secret key and an insecure channel, we introduce an intermediate system *URI (uniform random injection)* that provides the sender and receiver access to the same uniform random injections and their inverses chosen as follows: For each combination of N, A, λ, and message length ℓ, an injective function that maps strings of length ℓ to strings of length $\ell + \lambda$ is chosen uniformly at random.

As we shall see, this ideal resource can be constructed from a shared secret key using an RAE scheme in a straightforward manner. We then construct several channels from URI and an insecure channel. The advantage of this approach is that all further constructions based on URI are information-theoretic, i.e., we do not have to relate the security of each construction step to the RAE security game. Instead, we can rely on the composition theorem to guarantee the security of the overall constructions.

Random injection channel. We show that one can construct a channel we call *RIC (random injection channel)* from URI and an insecure channel by fixing λ and using a counter as the nonce. RIC can be seen as

a further intermediate step towards constructing ASC, that in addition allows us to analyze best-possible security.

The channel RIC takes as input a pair (A, m) from the sender and leaks A and the length of m to the adversary. The adversary can deliver the pair (A, m), and further at any point in time try to inject a new message of length ℓ and some value A. The probability with which such an injection is successful depends on λ and ℓ. In case of a success, an almost uniform message of length ℓ from the message space together with A is delivered to the receiver. If an injection was successful and the tuple (A, m) was received, and if the sender subsequently sends exactly the pair (A, m), then the adversary is notified about this repetition.

Best possible authenticity and confidentiality. If ASC is considered as the ultimate goal of RAE and authenticated encryption in general, the only shortcomings of RIC are that it is possible to inject messages with positive probability and that, if an attempted message injection was successful, the channel leaks a certain repetition to the adversary. While the first shortcoming is a lack of authenticity, the second one is a lack of confidentiality. While the type of leakage violating confidentiality might seem artificial, we describe an application in which such leakage might be problematic. Briefly, the leakage can reveal hidden information flow from the receiver to the sender.

We then analyze whether RAE really achieves the best-possible authenticity and confidentiality by bounding the probabilities of successful message injections and of leaking this particular repetition pattern for arbitrary schemes for achieving authenticity and confidentiality. While it is straightforward to see that authenticity requires redundancy and therefore a large ciphertext expansion, one might hope that the repetition leakage can be avoided. We prove that this is not the case, i.e., we show that the probability of an adversary being able to observe such a repetition is at least as high as in RIC, no matter what scheme is used or which setup assumptions are made.

To illustrate this lack of confidentiality for a concrete scheme, consider the following scenario in which the one-time pad is used over an insecure channel: Assume the attacker injects a ciphertext to Bob who decrypts it using the shared secret key and outputs the resulting message. Further assume Alice afterwards sends a message to Bob which results in the

attacker seeing the same ciphertext. In that case, the attacker learns the fact that the message sent by Alice equals the message output by Bob. This contradicts the understanding of confidentiality as revealing nothing except the length of the transmitted message.[1] Our results generalize this observation to arbitrary schemes. We thereby refine the understanding of what symmetric cryptography can and cannot achieve by showing that confidentiality, quite surprisingly, also requires redundancy in the ciphertexts when only insecure channels and an arbitrary setup are assumed, even if the protocol can keep state.

Augmented secure channels and message redundancy. Since the probability of successful message injections decreases exponentially with λ, RIC is indistinguishable from (a particular type of) ASC for large λ. We further provide a construction that incorporates an idea from [HKR15] to exploit the redundancy in messages to achieve a better bound. Our construction reveals the exact trade-off between ciphertext expansion and redundancy to achieve a required security level.

Nonce-reuse resistance. It was claimed in [HKR15] that reusing nonces only results in leaking the repetition pattern of messages, but does not compromise security beyond that. However, the claim was neither formalized nor proven. We fill this gap by introducing the channel resource *RASC (Repetition ASC)* that, aside of the length of each message, leaks the repetition pattern of the transmitted messages to the adversary. Furthermore, the adversary can deliver messages out-of-order and arbitrarily replay messages. We show that RASC can be constructed from URI and an insecure channel if the used nonce is always the same. This confirms the informal claim from [HKR15] and makes explicit that some authenticity is lost by allowing the adversary to reorder messages.

[1] This also contradicts a prior result in [MRT12] that claims that the one-time pad constructs a certain (fully) confidential channel, a so-called XOR-malleable channel, from an insecure channel and a shared key. The proof in that paper is flawed in that the simulation fails if more ciphertexts are injected than messages sent.

Resource **URI**

Initialization

for $(N, A, \lambda, \ell) \in \mathcal{N} \times \mathcal{H} \times \mathbb{N} \times \mathbb{N}$ **do**
  $f_{N,A,\lambda,\ell} \twoheadleftarrow \mathrm{Inj}\left(\Sigma^{\ell}, \Sigma^{\ell+\lambda}\right)$

Interface $\mathtt{X} \in \{\mathtt{A}, \mathtt{B}\}$

Input: $(\mathtt{fun}, N, A, \lambda, x) \in \mathcal{N} \times \mathcal{H} \times \mathbb{N} \times \mathcal{M}$
  output $f_{N,A,\lambda,|x|}(x)$ at $\mathtt{X}$

Input: $(\mathtt{inv}, N, A, \lambda, y) \in \mathcal{N} \times \mathcal{H} \times \mathbb{N} \times \mathcal{C}$
  output $f^{-1}_{N,A,\lambda,|y|-\lambda}(y)$ at $\mathtt{X}$

Figure 4.2: Uniform random injection resource. Interface E remains inactive.

4.2 Shared Uniform Random Injections

In this section, we describe the resource **URI** that grants access to shared uniform random injections and their inverses at interfaces A and B, and no access at interface E. We then use **URI** to define the security of RAE schemes and show that any RAE scheme that satisfies this definition can be used to construct **URI** from a shared secret key. Though syntactically different, it is easy to see that our definition is equivalent to the security definition from [HKR15].

We first give a definition for the uniform random injection system **URI**.

Definition 4.2.1. The resource **URI** has interfaces A, B, and E and takes inputs of the form $(\mathtt{fun}, N, A, \lambda, x)$ and $(\mathtt{inv}, N, A, \lambda, y)$ at interfaces A and B for $N \in \mathcal{N}$, $A \in \mathcal{H}$, $\lambda \in \mathbb{N}$, $x \in \mathcal{M}$, and $y \in \mathcal{C}$. Any input at interface E is ignored. We assume the domains $\mathcal{N}$, $\mathcal{H}$, $\mathcal{M}$, and $\mathcal{C}$ are equal to Σ^*. On input $(\mathtt{fun}, N, A, \lambda, x)$ at interface A or B, it returns $f_{N,A,\lambda,|x|}(x)$ at the same interface. On input $(\mathtt{inv}, N, A, \lambda, y)$, it returns $f^{-1}_{N,A,\lambda,|y|-\lambda}(y)$ if $|y| > \lambda$, and $\perp$ otherwise. The function $f_{N,A,\lambda,\ell}$ is chosen uniformly at random from the set $\mathrm{Inj}\left(\Sigma^{\ell}, \Sigma^{\ell+\lambda}\right)$ when needed for the first time and reused for later inputs. See Figure 4.2 for pseudocode.

4.2.1 Definition of RAE Security

We define a shared key resource $\mathbf{SK}_{\mathcal{K}}$ for some key distribution $\mathcal{K}$. The resource initially chooses a key according to $\mathcal{K}$ and outputs this key to interfaces A and B while interface E remains inactive, see Figure 4.3. Slightly abusing notation, we will also refer to the key space by $\mathcal{K}$ whenever no confusion can arise. We further define the converter rae_Π that is based on an RAE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. First, rae_Π requests the key from $\mathbf{SK}_{\mathcal{K}}$. For any input at the outer interface, it evaluates $\mathcal{E}$ or $\mathcal{D}$ using that key (and the arguments provided in the input) and returns the result. The code is given in Figure 4.3.

We consider an RAE scheme secure if all efficient distinguishers have poor advantage with respect to the following definition.

Definition 4.2.2. The advantage of a distinguisher $\mathbf{D}$ for an RAE scheme Π is quantified as

$$\mathsf{Adv}^{\mathsf{rae}}_{\Pi}(\mathbf{D}) \coloneqq \Delta^{\mathbf{D}}(\mathsf{rae}_\Pi{}^{\mathtt{A}}\, \mathsf{rae}_\Pi{}^{\mathtt{B}}\, \mathbf{SK}_{\mathcal{K}}, \mathbf{URI}).$$

It is straightforward to see that the definition implies the following construction statement according to Definition 2.3.3 in Section 2.3.4. Recall from Section 2.3 that we denote by $\mathsf{0}$ the converter that blocks any interaction at the interface to which it is attached.

Lemma 4.2.3. *The protocol* $(\mathsf{rae}_\Pi, \mathsf{rae}_\Pi)$ *constructs* $\mathbf{URI}$ *from* $\mathbf{SK}_{\mathcal{K}}$ *within* $\mathsf{Adv}^{\mathsf{rae}}_{\Pi}$ *with respect to simulator* $\mathsf{sim} := \mathsf{0}$ *and the pair* $(\mathsf{0}, \mathsf{0})$ *(i.e., in the Alice-Bob-Eve setting where Eve is inactive).*

Proof. Since interface E of $\mathbf{SK}_{\mathcal{K}}$ and $\mathbf{URI}$ are inactive, the converter $\mathsf{0}$ has no effect when connected to that interface, i.e., $\mathsf{0}^{\mathtt{E}}\, \mathbf{SK}_{\mathcal{K}} = \mathbf{SK}_{\mathcal{K}}$ and $\mathsf{0}^{\mathtt{E}}\, \mathbf{URI} = \mathsf{sim}^{\mathtt{E}}\, \mathbf{URI} = \mathbf{URI}$. Thus, both the correctness and the security condition of the construction are equivalent to

$$\Delta^{\mathbf{D}}(\mathsf{rae}_\Pi{}^{\mathtt{A}}\, \mathsf{rae}_\Pi{}^{\mathtt{B}}\, \mathbf{SK}_{\mathcal{K}}, \mathbf{URI}) \leq \mathsf{Adv}^{\mathsf{rae}}_{\Pi}(\mathbf{D})$$

for all distinguishers $\mathbf{D}$, which trivially holds by definition of $\mathsf{Adv}^{\mathsf{rae}}_{\Pi}$. □

To conclude this section on the alternative definition, we show that Definition 4.2.2 and Definition 4.1.2 are equivalent:

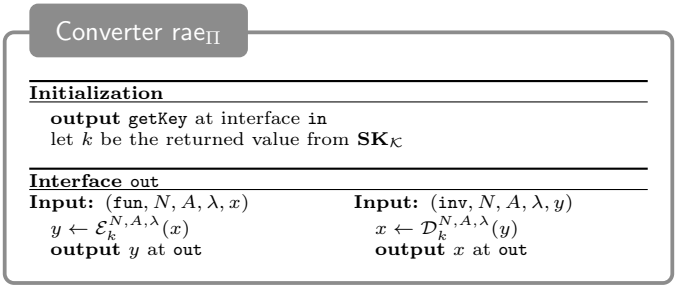


Figure 4.3: Protocol that constructs **URI** from a shared secret key.

Lemma 4.2.4. *For every distinguisher* $\mathbf{D}$ *there is an adversary* $\mathcal{A}$ *(with essentially the same efficiency) such that* $\mathsf{Adv}_\Pi^{\textit{rae-game}}(\mathcal{A}) = \mathsf{Adv}_\Pi^{\textit{rae}}(\mathbf{D})$. *Conversely, for every adversary* $\mathcal{A}$ *there is a distinguisher* $\mathbf{D}$ *(with essentially the same efficiency) such that* $\mathsf{Adv}_\Pi^{\textit{rae}}(\mathbf{D}) = \mathsf{Adv}_\Pi^{\textit{rae-game}}(\mathcal{A})$.

Proof. Observe that the inputs $(\mathtt{fun}, \cdot, \cdot, \cdot, \cdot)$ and $(\mathtt{inv}, \cdot, \cdot, \cdot, \cdot)$ to system $\mathsf{rae}_\Pi{}^{\mathsf{A}}\,\mathsf{rae}_\Pi{}^{\mathsf{B}}\,\mathbf{SK}_\mathcal{K}$ correspond to queries in $\mathbf{Real}_{\mathbf{\Pi}}^{\mathrm{rae}}$ to the oracles `Enc` and `Dec`, respectively: in both cases, the resulting outputs are generated by the algorithms $\mathcal{E}$ and $\mathcal{D}$ for a uniformly random key. Similarly, the same inputs to the resource **URI** correspond to the oracles `Enc` and `Dec` in the game $\mathbf{RAE}_\mathbf{\Pi}$ since the outputs are in both cases computed by a uniformly random injection or its inverse. Hence, a distinguisher can be turned into an adversary with the same advantage and vice versa by exchanging inputs to the resources with the corresponding oracle queries. □

4.3 Random Injection Channels

The goal of the current section is to examine the exact security achieved by RAE schemes when used to protect communication. We present constructions of specific secure channels from insecure channels and resource **URI** where each type of secure channel precisely captures the amount of leakage to an eavesdropper and the possible influence of an adversary interfering with the protocol execution. This new resource formalizes security for any expansion. As an additional result, we are able to answer what best-possible communication security is and observe that RAE

schemes achieve this level of security.

The constructed channel. The channel we construct in this section is defined in Figure 4.4 and can be roughly described as follows: It allows to repeatedly send pairs (A_i, m_i) in an ordered fashion from a sender to a receiver. Each pair consists of the associated data A_i and the message m_i. The attacker is limited to seeing the associated data A_i and the length of the message $|m_i|$ of each transmitted pair. Additionally, the attacker learns whether the ith injected pair equals the one that is currently sent.

The attacker can either deliver the next legitimate pair (A_i, m_i) or try to inject a pair (A, m) that is different from (A_i, m_i). Such an injection is only successful with a certain probability. The associated data A and the length ℓ of the message are chosen by the attacker and m is a uniformly random message of length ℓ if $A \neq A_i$. Otherwise, m is a uniformly random message $m \neq m_i$ of length ℓ. If an injection attempt is not successful, the resource does not deliver messages at interface `B` any more and signals an error by outputting $\perp$. If the adversary injects the ith message, the legitimate ith message cannot be delivered anymore.[2]

The success probability of an injection attempt depends on the expansion λ and the specified message length ℓ and whether the sender's queue $\mathcal{S}$ is empty or not. The exact probabilities are quantified by the two sampling functions Sample and SamplExcl. The function Sample first samples a bit according to the probability that a fixed element from $\Sigma^{\ell+\lambda}$ has a preimage under a uniform random injection $\Sigma^{\ell} \to \Sigma^{\ell+\lambda}$. If the bit is 1, a uniform random preimage is returned. The function SampleExcl essentially does the same, but the domain and codomain are both reduced by one element.[3]

4.3.1 Constructing Random Injection Channels

We construct resource $\mathbf{RIC}_\lambda$ from $[\mathbf{URI}, \mathbf{IC}]$ which denotes the resource that provides at each interface access to the corresponding interface of

[2] This relates to the security of RAE schemes which ensures that the message cannot be decrypted using a wrong nonce. In our construction, the nonce is implemented as the sequence number.

[3] This ensures that the injected message is different from the one that the sender provided.

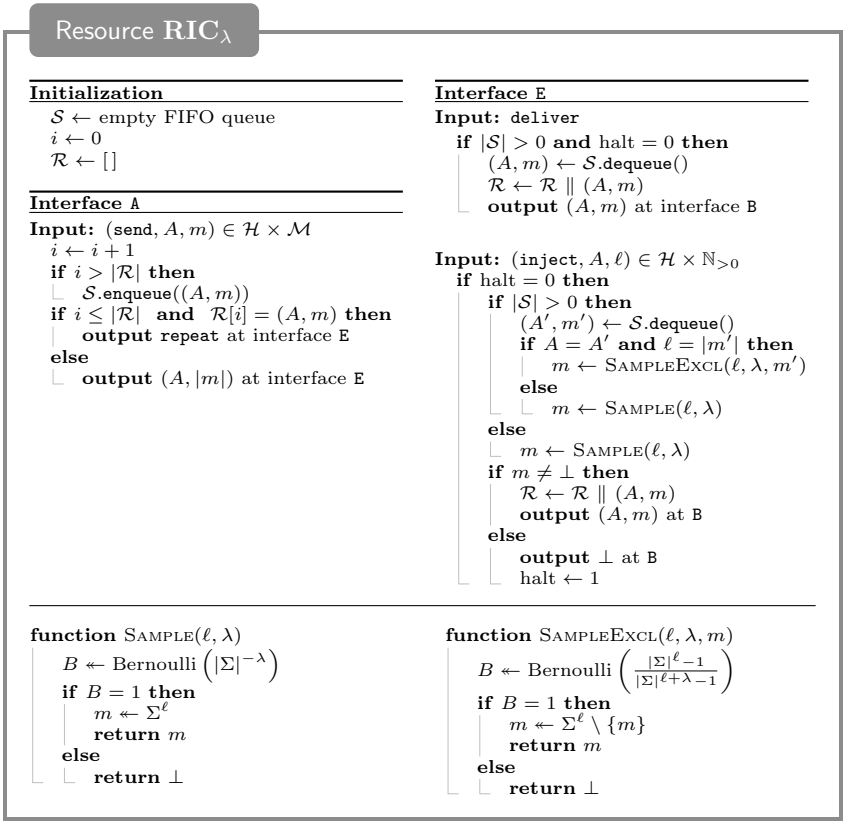


Figure 4.4: Description of $\mathbf{RIC}_\lambda$. In the description, Bernoulli(p) denotes the distribution over $\{0, 1\}$, where 1 has probability p and 0 has probability $1 - p$.

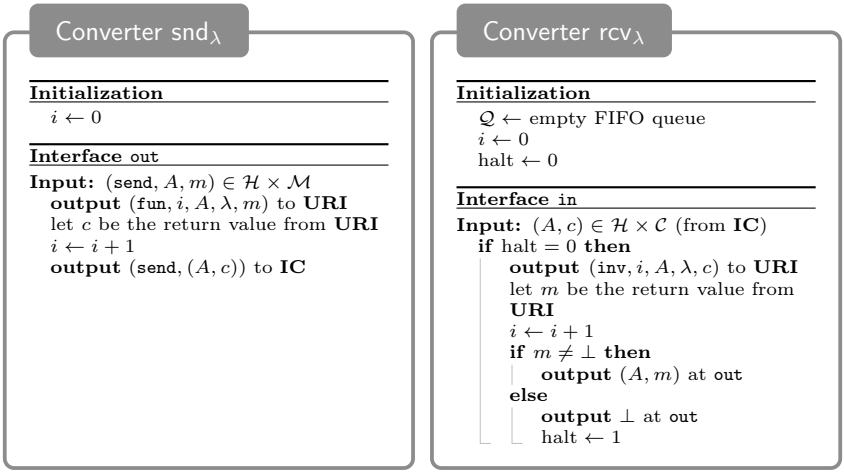


Figure 4.5: The converters for the sender (left) and the receiver (right) to construct $\mathbf{RIC}_\lambda$.

both resources. Our protocol specifies a particular but very natural usage of **URI** where the nonce is implemented as a counter value.[4]

Protocol

We present the protocol as pseudocode in Figure 4.5. The converter for the sender, snd_λ, accepts inputs of the form $(\mathtt{send}, A, m)$ at its outer interface. It outputs $(\mathtt{fun}, i, A, \lambda, m)$ at the inner interface to **URI**. The nonce is implemented as a counter and λ is the parameter of the protocol. Once a ciphertext is received as a return value from **URI**, it is output together with its associated data at the inner interface for the insecure channel **IC**. The receiver converter rcv_λ receives ciphertexts together with the associated data at its inner interface from **IC** and decrypts c using parameters A, i and λ. On success, the corresponding plaintext is output at the outer interface. If decryption fails, the converter stops and signals an error by outputting $\perp$.

[4]Implementing the nonce as a counter allows to maintain the order of messages.

Security Proof

In order to show that the protocol $(\mathsf{snd}_\lambda, \mathsf{rcv}_\lambda)$ constructs $\mathbf{RIC}_\lambda$ from $[\mathbf{URI}, \mathbf{IC}]$, we prove again the correctness and security conditions as outlined in Sections 2.3.4 and 2.3.5. For all channels, the converter that formalizes the default honest behavior at interface $\mathtt{E}$ is the converter dlv introduced in Section 2.3.5.

Theorem 4.3.1. *Let $\lambda \in \mathbb{N}$. The protocol $(\mathsf{snd}_\lambda, \mathsf{rcv}_\lambda)$ constructs resource $\mathbf{RIC}_\lambda$ from $[\mathbf{URI}, \mathbf{IC}]$ with respect to $(\mathsf{dlv}, \mathsf{dlv})$ and simulator $\mathsf{sim}_{\mathrm{RIC}}$ as defined in Figure 4.6. More specifically, for all distinguishers $\mathbf{D}$*

$$\Delta^{\mathbf{D}}(\mathsf{snd}_\lambda^{\mathtt{A}}\mathsf{rcv}_\lambda^{\mathtt{B}}\mathsf{dlv}^{\mathtt{E}}[\mathbf{URI}, \mathbf{IC}], \mathsf{dlv}^{\mathtt{E}}\mathbf{RIC}_\lambda) = 0 \tag{4.1}$$

and

$$\Delta^{\mathbf{D}}(\mathsf{snd}_\lambda^{\mathtt{A}}\mathsf{rcv}_\lambda^{\mathtt{B}}[\mathbf{URI}, \mathbf{IC}], \mathsf{sim}_{\mathrm{RIC}}^{\mathtt{E}}\mathbf{RIC}_\lambda) = 0. \tag{4.2}$$

Proof. We first prove the security condition (4.2) by analyzing the input-output behavior of both systems involved. To this end, we consider the possible inputs at each interface.

On the ith input $(\mathtt{send}, A_i, m_i)$ at interface $\mathtt{A}$: In $\mathsf{snd}_\lambda^{\mathtt{A}}\mathsf{rcv}_\lambda^{\mathtt{B}}[\mathbf{URI}, \mathbf{IC}]$, converter snd_λ evaluates $\mathbf{URI}$ on input (i, A_i, λ, m_i), where the counter i is the nonce and A_i is the associated data. The associated data is sent together with $\mathbf{URI}$'s return value c_i over $\mathbf{IC}$. If this is the first query to $\mathbf{URI}$ with parameters i and A_i, then the output c_i is distributed uniformly at random over $\Sigma^{|m_i|+\lambda}$. If there has been a query $(\mathtt{inv}, i, A_i, c_i')$ before at interface $\mathtt{B}$ of $\mathbf{URI}$, with $|c_i'| = |m_i| + \lambda$ that has generated the output $m_i' = m_i$, then c_i is determined to be equal to c_i'. If $m_i \neq m_i'$, the output is c_i is distributed uniformly at random over $\Sigma^{|m_i|+\lambda} \setminus \{c_i'\}$.

In system $\mathsf{sim}_{\mathrm{RIC}}^{\mathtt{E}}\mathbf{RIC}_\lambda$, (A_i, m_i) is inserted into the senders queue of $\mathbf{RIC}_\lambda$. If there has already been an ith output (A_i', m_i') with $A_i' = A_i$ and $m_i = m_i'$ at interface $\mathtt{B}$, then $\mathbf{RIC}_\lambda$ outputs $\mathtt{repeat}$ at interface $\mathtt{E}$ and the simulator outputs the ciphertext and the associated data that was input before at its outer interface as the ith injection. Otherwise, $\mathbf{RIC}$ outputs the pair $(A_i, |m_i|)$ to $\mathsf{sim}_{\mathrm{RIC}}$. The simulator checks whether there exists an ith injected associated data-ciphertext pair (A_i', c_i') with $A_i' = A_i$ and $|c'| = |m_i| + \lambda$. If such a pair exists, which is the case if $\mathrm{diff} > 0$, $\mathsf{sim}_{\mathrm{RIC}}$ generates a

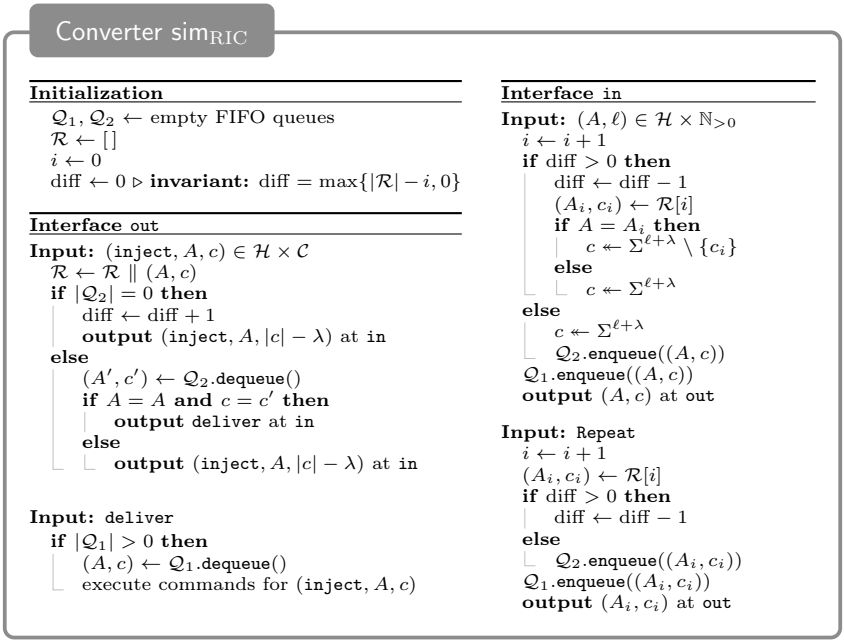


Figure 4.6: Simulator for the security condition of the construction of $\mathbf{RIC}_\lambda$.

uniformly random string $c_i \in \Sigma^{|m_i|+\lambda} \setminus \{c_i'\}$ and outputs (A_i, c_i) at its outer interface. If there is no such pair, i.e., if diff $= 0$, $\mathsf{sim}_{\mathrm{RIC}}$ generates a uniformly random string $c_i \in \Sigma^{|m_i|+\lambda}$ and outputs (A_i, c_i) at its outer interface. In both cases, $\mathsf{sim}_{\mathrm{RIC}}$ stores (A_i, c_i) in its own queues for later reference. $\mathcal{Q}_1$ simulates the queue of the insecure channel in the real world (to correctly simulate deliver-queries). $\mathcal{Q}_2$ stores the associated data-ciphertext pairs of the corresponding entries in the queue $\mathcal{S}$ of $\mathbf{RIC}_\lambda$. We observe that the case distinctions made by $\mathsf{sim}_{\mathrm{RIC}}$ correspond exactly to the cases that happen in the real system. Hence, the output distribution at interface E is identical to above.

On the ith input $(\mathtt{inject}, A_i, c_i)$ **at interface** E**:** In the following, we

assume that interface B has not output $\perp$ already, as otherwise the two systems behave identically anyway.

In $\mathsf{snd}^{\mathtt{A}}_{\lambda}\mathsf{rcv}^{\mathtt{B}}_{\lambda}[\mathbf{URI}, \mathbf{IC}]$, the converter rcv_{λ} queries **URI** with input $(\mathtt{inv}, i, A_i, \lambda, c_i)$ to receive the preimage m_i of c_i under parameters (i, A_i, λ). Let us first assume that there has been an ith input before at interface A (A'_i, m'_i) and that **URI** generated the ciphertext c'_i on that input. There are three cases to consider to determine the output distribution at interface B:

1. If $A_i = A'_i$ and $c_i = c'_i$, then $m_i = m'_i$ holds with probability 1 and dec_{Π} outputs m'_i.
2. If $A_i \neq A'_i$ or $|c_i| \neq |c'_i|$, **URI** generates a fresh uniform injection $\Sigma^{|c_i|-\lambda} \to \Sigma^{|c_i|}$ and there is only an output at interface B if c_i has a preimage under the chosen injection. This is the case with probability

$$\frac{|\Sigma^{|c_i|-\lambda}|}{|\Sigma^{|c_i|}|} = |\Sigma|^{-\lambda}.$$

   Given that c_i has a preimage, then the output at interface B is (A_i, m_i), where m_i is distributed uniformly over $\Sigma^{|c_i|-\lambda}$, since all preimages are equally likely. If c_i does not have a preimage, dec_{Π} sets halt $\leftarrow 1$ and outputs $\perp$.
3. If $A_i = A'_i$, $c_i \neq c'_i$, and $|c_i| = |c'_i|$, **URI** has already generated an injection $f\colon \Sigma^{|c_i|-\lambda} \to \Sigma^{|c_i|}$ on the ith input (A'_i, m'_i) at interface A such that $f(m'_i) = c'_i$, and there is only an output at interface B if c_i has a preimage under f. Since there are $|\Sigma^{|c_i|-\lambda} \cap \mathcal{M}_v| - 1$ possible preimages different from m'_i and each of them is mapped to one out of $|\Sigma^{|c_i|}| - 1$ possible ciphertexts different from c'_i, the probability that c_i has a preimage is

$$\frac{|\Sigma^{|c_i|-\lambda}| - 1}{|\Sigma^{|c_i|}| - 1}.$$

   If c_i has a preimage, then the output at interface B is (A_i, m_i), where m_i is distributed uniformly over $\Sigma^{|c_i|-\lambda} \setminus \{m'_i\}$, where m'_i is excluded due to injectivity and the remaining preimages are equally likely. If c_i does not have a preimage, dec_{Π} sets halt $\leftarrow 1$ and outputs $\perp$.

If there has not yet been an ith input (A'_i, m'_i) at interface $\mathtt{A}$ (i.e., there are more messages injected than sent), the behavior is the same as in Item 2 above, because **URI** generates a fresh uniform injection $\Sigma^{|c_i|-\lambda} \to \Sigma^{|c_i|}$ for each counter value i.

In $\mathsf{sim}^{\mathtt{E}}_{\mathrm{RIC}}\mathbf{RIC}_\lambda$, let us again first assume that there are more messages sent than injected, i.e., $\mathrm{diff} = 0$. In that case, $\mathsf{sim}_{\mathrm{RIC}}$ retrieves the front element of $\mathcal{Q}_2$ which contains the simulated pair (A'_i, c'_i) of the front element of $\mathcal{S}$, say (A'_i, m'_i) (where m'_i is the next message that can be delivered reliably). If $A_i = A'_i$ and $c_i = c'_i$ then $\mathsf{sim}_{\mathrm{RIC}}$ delivers the next element in the sender queue of $\mathbf{RIC}_\lambda$ and hence the ith message m'_i is output. This corresponds to Item 1 above. In the other two cases, $\mathsf{sim}_{\mathrm{RIC}}$ outputs $(\mathtt{inject}, A, |c_i| - \lambda)$. The behavior of $\mathbf{RIC}_\lambda$ is then as follows: Let $\ell := |c_i| - \lambda$. If $\ell = |m'_i|$ and $A_i = A'_i$, the output is $\bot$ with probability $1 - \frac{|\Sigma|^\ell - 1}{|\Sigma|^{\ell+\lambda} - 1}$. If the output is not $\bot$, the message is chosen uniformly at random from $\Sigma^\ell \setminus \{m_i\}$. This behavior follows from the call to the function $\textsc{SampleExcl}(\ell, \lambda, m_i)$. If $\ell \neq |m_i|$ or $A_i \neq A_i$, $\mathbf{RIC}_\lambda$ either outputs $\bot$ and halts with probability $1 - |\Sigma|^{-\lambda}$ or, conditioned on not being $\bot$, the output at interface $\mathtt{B}$ is (A_i, m_i), where m_i is chosen uniformly at random from Σ^ℓ. This behavior follows from the call to the function $\textsc{Sample}(\ell, \lambda)$.

Finally, if there is no element in the sender's queue $\mathcal{S}$, i.e., if there are at least as many messages injected as sent, then the output distribution is identical to the case $\ell \neq |m_i|$ or $A_i \neq A_i$ like in the real world.

We conclude that in any case, the outputs are distributed identically for the systems $\mathsf{snd}^{\mathtt{A}}_\lambda \mathsf{rcv}^{\mathtt{B}}_\lambda[\mathbf{URI}, \mathbf{IC}]$ and $\mathsf{sim}^{\mathtt{E}}_{\mathrm{RIC}}\mathbf{RIC}_\lambda$.

On the ith input (`deliver`) at interface `E`: In $\mathsf{snd}^{\mathtt{A}}_\lambda \mathsf{rcv}^{\mathtt{B}}_\lambda[\mathbf{URI}, \mathbf{IC}]$, the front element of the sender queue (within **IC**) is decrypted by rcv_λ. This behavior is perfectly simulated in $\mathsf{sim}^{\mathtt{E}}_{\mathrm{RIC}}\mathbf{RIC}_\lambda$ since the simulator retrieves the front element (E, c) of its queue $\mathcal{Q}_1$ which simulates the real-world sender queue. Next, the simulator executes the same instructions as for $(\mathtt{inject}, E, c)$ which lets the two systems produce identically distributed outputs.

This concludes the analysis of the security condition.

We now prove condition (4.1). First, in $\mathsf{enc}_\Pi{}^{\mathtt{A}}\mathsf{dec}_\Pi{}^{\mathtt{B}}\mathsf{dlv}^{\mathtt{E}}[\mathbf{URI}, \mathbf{IC}]$, the converter dlv is attached at interface E and answers any output produced by **IC** with the input (deliver). This essentially converts **IC** into a reliable transmission channel: whatever pair (A, c) is given to **IC**, it is immediately delivered to dec_Π. Therefore, if the ith input at interface A is $(\mathtt{send}, A_i, m_i)$ then the ith output at interface B is (A_i, m_i), since **URI** is queried with the exact same parameters. It is obvious that the same holds for the input-output behavior of system $\mathsf{dlv}^{\mathtt{E}}\mathbf{RIC}_\lambda$. □

4.4 What is Best-Possible Security?

We observe that $\mathbf{RIC}_\lambda$ has two undesirable properties: messages can be injected and the output at interface E leaks more than only the length of the payload in that it reveals whether Alice sends the pair (A, m) that has been output by Bob upon an adversarial injection. In contrast, a channel that only leaks the message length is considered fully confidential.

We first illustrate an application in which this lack of full confidentiality is problematic. The main purpose of our example is to show that one cannot exclude the existence of an application where exactly this (intuitively small) difference to full confidentiality yields a security problem.

Second, we show that a successful injection followed by the undesired information leakage about the repetition is possible for any scheme, even if it is stateful and uses an arbitrary setup before starting communication, and that the probability of this is minimized if $\mathbf{RIC}_\lambda$ is used.

Sample scenario: On the difference to full confidentiality. Assume a setting in which party A is allowed to send information to party B via a fully confidential channel but not vice versa. Suppose now that B finds a possibility to send information to A via a covert channel and the two parties use the confidential channel for messages from A to B and the covert channel for messages from B to A. Suppose now that the confidential channel is in fact a channel that leaks the above repetition event instead of only the message length. This gives an investigator E a means to test for the existence of a covert channel from B to A as follows: At some point, E injects a random message m to B. Assuming information flow from B to A, party B might start a discussion about m with party A.

As part of this conversation, A might send m to B, which would signal a repetition-event to E. For large message spaces, it is very unlikely that A comes up with the exact same message that was randomly injected to B before, unless there is a (hidden) flow of information. The occurrence of the event is therefore a witness for the existence of a channel from B to A. In contrast, a fully confidential channel would not reveal the existence of the covert channel.

4.4.1 RIC Characterizes Best-Possible Security

In $\mathbf{RIC}_\lambda$, an injection attempt is successful with probability at most $\frac{|\mathcal{M}|}{|\mathcal{C}|} = |\Sigma|^{-\lambda}$ and given a successful injection and that Alice subsequently sends the corresponding output of Bob, the above described leakage occurs with probability 1. Overall, the total probability that an undesired property can be observed is bounded by $|\Sigma|^{-\lambda}$.

We show that this probability is optimal and that no protocol can achieve a better bound. Hence, $\mathbf{RIC}_\lambda$ maximizes authenticity and confidentiality. We first prove the following general lemma.

Lemma 4.4.1. *Let $\mathcal{M}$ and $\mathcal{C}$ be finite nonempty sets and let E and D be random variables over functions $\mathcal{H} \times \mathcal{M} \to \mathcal{C}$ and $\mathcal{H} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$, respectively, such that*

$$\forall m \in \mathcal{M}, a \in \mathcal{H}: \quad \Pr[D(a, E(a, m)) = m] \geq p$$

for some $p \in [0, 1]$. We then have for all $a \in \mathcal{H}$ and any random variable C that is distributed uniformly over $\mathcal{C}$ and independent from E and D,

$$\Pr[D(a, C) \neq \perp \wedge E(a, D(a, C)) = C] \geq p \cdot \frac{|\mathcal{M}|}{|\mathcal{C}|}.$$

Proof. We have for all $a \in \mathcal{H}$

$$\begin{aligned}
&\Pr[D(a,C) \neq \bot \wedge E(a,D(a,C)) = C] \\
&\qquad = \sum_{m \in \mathcal{M}} \sum_{c \in \mathcal{C}} \Pr[D(a,c) = m \wedge E(a,m) = c \wedge C = c] \\
&\qquad = \frac{1}{|\mathcal{C}|} \sum_{m \in \mathcal{M}} \underbrace{\sum_{c \in \mathcal{C}} \Pr[D(a,c) = m \wedge E(a,m) = c]}_{= \Pr[D(a,E(a,m)) = m] \;\geq\; p} \\
&\qquad \geq\; p \cdot \frac{|\mathcal{M}|}{|\mathcal{C}|},
\end{aligned}$$

where we used in the second step that C is distributed uniformly over $\mathcal{C}$ and independent from E and D. □

Lemma 4.4.1 can be applied to our usual setting $\mathsf{enc}_\lambda^{\mathsf{A}}\mathsf{dec}_\lambda^{\mathsf{B}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]$ for a generic protocol $(\mathsf{enc}_\lambda, \mathsf{dec}_\lambda)$ in a straightforward manner: we only have to observe that for the ith input $(\mathtt{send}, A_i, m_i)$, for all $i \in \mathbb{N}$, converter enc_λ is characterized by a probabilistic map $\mathcal{H} \times \mathcal{M} \to \mathcal{C}$, that may depend on previous inputs and outputs and on the key k. Similarly, the converter dec_λ is characterized by a probabilistic map $\mathcal{H} \times \mathcal{C} \to \mathcal{M} \cup \{\bot\}$ for any $i \in \mathbb{N}$.

Correctness of the protocol implies that if $(\mathtt{send}, A_i, m_i)$ is input to enc_λ as the ith query and yields ciphertext c_i, then the probability that on the ith input (A_i, c_i) to dec_λ, and if dec_λ has not halted yet, the converter decrypts the ciphertext to m_i with probability p; note that $p = 1$ for RAE schemes. Hence, Lemma 4.4.1 implies that the probability that any of the two undesirable properties can be observed during protocol execution is at least $|\Sigma|^{-\lambda}$.

4.5 Further Applications of the New Concept

We conclude our study by completing two security claims stated in [HKR15] without a proof. First, we show how concretely redundancy in messages can amplify security and second, we show in what exact sense RAE is nonce-reuse resistant. The reader might recall Section 3.2.4 and in particular the definition of the channel resource $\mathbf{SEC}_{\mathrm{TLS}}$ in Figure 3.5 before reading the rest of this chapter.

4.5.1 Security by Verifiable Redundancy

Looking at the specification of $\mathbf{RIC}_\lambda$, we observe that for large values λ, the probability of successful injections becomes exponentially small, and so are the repetition events at interface $\mathtt{E}$. It is now easy to derive the description of a channel without the two shortcomings: it is exactly the channel $\mathbf{SEC}_{\mathrm{TLS}}$ we introduced in Figure 3.5 in Chapter 3, which is a particular type of an ASC.

We formally show how to construct this channel and how redundancy in messages can be exploited to improve authenticity, where redundancy restricts the set of valid messages to a subset of $\mathcal{M} = \Sigma^*$. The following theorem provides the exact security bound in terms of redundancy in the message space and ciphertext expansion λ. We thereby confirm a conjecture of [HKR15]. Let $v : \mathcal{M} \mapsto \{\mathtt{true}, \mathtt{false}\}$ be a predicate on the message space. We define the subset $\mathcal{M}_v \coloneqq \{m \mid m \in \mathcal{M} \ \wedge \ v(m)\}$ which we call the set of valid messages. Following [HKR15], the density of $\mathcal{M}_v$ is defined as

$$d_v \coloneqq \max_{\ell \in \mathbb{N}} \frac{|\Sigma^\ell \cap \mathcal{M}_v|}{|\Sigma^\ell|}.$$

The constructed channel. Formally, we consider the channel $\mathbf{SEC}_{\mathrm{TLS}}$ as in Figure 3.5 with special message space $\mathcal{M} \coloneqq \mathcal{M}_v$ and special type (or AD) domain $\mathcal{T} \coloneqq \mathcal{H}$. Put differently, the channel is derived from $\mathbf{RIC}_\lambda$ by requiring that $m \in \mathcal{M}_v$ and by removing undesired capabilities that vanish due to the exponentially small success probability for large λ.

Protocol. The protocol for the sender, sndChk_v, accepts inputs of the form $(\mathtt{send}, A, m)$ at its outer interface and forwards the pair to the channel $\mathbf{RIC}_\lambda$ if and only if $v(m)$ (and otherwise ignores the request). The receiver converter $\mathsf{rcvChk}_\mathsf{v}$, on receiving the pair (A, m) from $\mathbf{RIC}_\lambda$, outputs (A, m) at its outer interface if and only if $v(m)$. If $\mathsf{rcvChk}_\mathsf{v}$ receives $\perp$ from $\mathbf{RIC}_\lambda$ or if $\neg v(m)$, it outputs $\perp$ at its outer interface and halts.

Theorem 4.5.1. *Let $\lambda \in \mathbb{N}$. The protocol $(\mathsf{sndChk}_v, \mathsf{rcvChk}_\mathsf{v})$ constructs $\mathbf{SEC}_{\mathrm{TLS}}$ (with message space $\mathcal{M}_v$ and type space $\mathcal{H}$) from $\mathbf{RIC}_\lambda$ with respect to $(\mathsf{dlv}, \mathsf{dlv})$ and simulator sim defined in Figure 4.7. More specifically,*

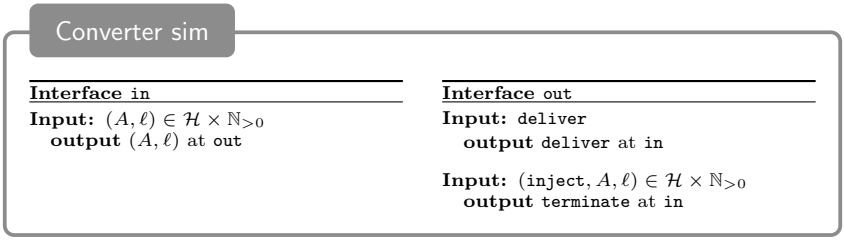


Figure 4.7: Simulator for the security condition of the construction of $\mathbf{SEC}_{\mathrm{TLS}}$ from $\mathbf{RIC}_\lambda$.

for all distinguishers $\mathbf{D}$

$$\Delta^{\mathbf{D}}(\mathsf{sndChk}_v^{\mathtt{A}}\mathsf{rcvChk}_{\mathsf{v}}{}^{\mathtt{B}}\mathsf{dlv}^{\mathtt{E}}\mathbf{RIC}_\lambda, \mathsf{dlv}^{\mathtt{E}}\mathbf{SEC}_{\mathrm{TLS}}) = 0 \tag{4.3}$$

$$\textit{and} \quad \Delta^{\mathbf{D}}(\mathsf{sndChk}_v^{\mathtt{A}}\mathsf{rcvChk}_{\mathsf{v}}{}^{\mathtt{B}}\mathbf{RIC}_\lambda, \mathsf{sim}^{\mathtt{E}}\mathbf{SEC}_{\mathrm{TLS}}) \leq d_v \cdot |\Sigma|^{-\lambda}. \tag{4.4}$$

Proof. The correctness condition (4.3) is straightforward to verify. We only prove the security condition (4.4). It is easy to see that the two systems behave identically as long as no injection attempt is successful. This is because successful injections are necessary for observing `repeat`: as long as no injection is successful, for any `send`-query to $\mathbf{RIC}_\lambda$, the condition $i \leq |\mathcal{R}|$ is not satisfied after incrementing i. We thus only have to bound this probability. We hence consider the event that in an interaction of a distinguisher with the real system $\mathsf{sndChk}_v^{\mathtt{A}}\mathsf{rcvChk}_{\mathsf{v}}{}^{\mathtt{B}}\mathbf{RIC}_\lambda$ the first attempt to inject a random message is successful (since in case of an unsuccessful attempt, both channels stop delivering messages). In any interaction of $\mathbf{D}$ with the resource, the probability of the event is determined by $\mathbf{RIC}_\lambda$ as one out of two possibilities, see Figure 4.4. For any $i \in \mathbb{N}$, if the ith query at interface `E` is the first attempt to inject a message, then the probability depends on whether the specified associated data and the length coincides with the length of the message and the associated data of the ith input at interface `A`. Both probabilities are

upper bounded by

$$\max\left\{\frac{|\Sigma^{|c_i|-\lambda}\cap\mathcal{M}_v|-1}{|\Sigma^{|c_i|}|-1},\frac{|\Sigma^{|c_i|-\lambda}\cap\mathcal{M}_v|}{|\Sigma^{|c_i|}|}\right\}\leq\frac{|\Sigma^{|c_i|-\lambda}\cap\mathcal{M}_v|}{|\Sigma^{|c_i|}|}$$
$$=\frac{|\Sigma^{|c_i|-\lambda}\cap\mathcal{M}_v|}{|\Sigma^{|c_i|-\lambda}|}\cdot|\Sigma|^{-\lambda}\leq d_v\cdot|\Sigma|^{-\lambda},$$

where we used $\frac{x-1}{y-1}\leq\frac{x}{y}$ for $x\leq y$ in the first step, and the definition of d_v in the last step. □

4.5.2 Guarantees for Nonce-Reuse

One goal of robust authenticated encryption is to provide resilience to the misuse when nonces are repeated. While the expected security loss was only informally stated in [HKR15], we rigorously derive the exact guarantees that can still be expected in such a scenario. To this end, we consider the extreme case where the nonce is a constant value.

Secure Channel with Repetition Leakage. The channel that is achieved if the nonce is repeating is denoted $\mathbf{RSEC}_{\mathrm{TLS}}$ and its description is given in Figure 4.8. There are two differences to $\mathbf{SEC}_{\mathrm{TLS}}$: First, not only the length of the message is leaked at interface $\mathtt{E}$ but also the number i of the first transmitted message that equals the current message. This leaks the repetition pattern of transmitted values. Second, the adversary can replay messages and induce arbitrary out-of-order delivery.

Protocol. The protocol, which we denote by $(\mathsf{rsnd},\mathsf{rrcv})$, invokes **URI** using the constant nonce 0. Furthermore, the protocol verifies that all messages are from the set $\mathcal{M}_v$. The protocol is specified in Figure 4.9.

Theorem 4.5.2. *Let $\lambda\in\mathbb{N}$. The protocol $(\mathsf{rsnd}_\lambda,\mathsf{rrcv}_\lambda)$ constructs $\mathbf{RSEC}_{\mathrm{TLS}}$ (with message space $\mathcal{M}_v$ and type space $\mathcal{H}$) from $[\mathbf{URI},\mathbf{IC}]$ with respect to $(\mathsf{dlv},\mathsf{dlv})$ and simulator sim defined in Figure 4.10. More specifically, we have for all distinguishers $\mathbf{D}$*

$$\Delta^{\mathbf{D}}(\mathsf{rsnd}_\lambda^{\mathtt{A}}\mathsf{rrcv}_\lambda^{\mathtt{B}}\mathsf{dlv}^{\mathtt{E}}[\mathbf{URI},\mathbf{IC}],\mathsf{dlv}^{\mathtt{E}}\mathbf{RSEC}_{\mathrm{TLS}})\leq\frac{2d_v+q\cdot(q-1)}{2}\cdot|\Sigma|^{-\lambda}\tag{4.5}$$

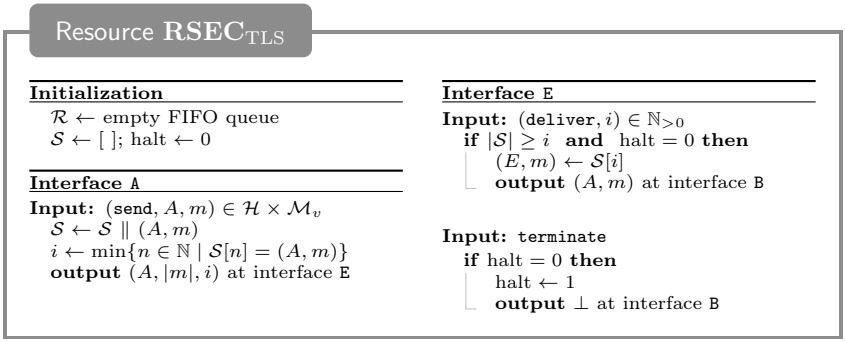


Figure 4.8: Description of $\mathbf{RSEC}_{\mathrm{TLS}}$.

and

$$\Delta^{\mathbf{D}}(\mathsf{rsnd}^{\mathtt{A}}_{\lambda}\mathsf{rrcv}^{\mathtt{B}}_{\lambda}[\mathbf{URI}, \mathbf{IC}], \mathsf{sim}^{\mathtt{E}}\mathbf{RSEC}_{\mathrm{TLS}}) \leq \frac{2d_v + q \cdot (q-1)}{2} \cdot |\Sigma|^{-\lambda}, \tag{4.6}$$

where q is the total number of inputs made by $\mathbf{D}$.

Proof. We prove the security condition (4.6) in a similar way as in the previous section.

On the ith input $(\mathtt{send}, A_i, m_i)$ at interface A: In $\mathsf{rsnd}^{\mathtt{A}}_{\lambda}\mathsf{rrcv}^{\mathtt{B}}_{\lambda}[\mathbf{URI}, \mathbf{IC}]$, system **URI** is queried with $(\mathtt{fun}, 0, A_i, \lambda, m_i)$ to produce ciphertext c_i. Next, the pair (A_i, c_i) is sent over **IC** and thus output at interface E. Clearly, if there exists some $j < i$ s.t. $A_j = A_i$ and $m_j = m_i$ then $c_j = c_i$ (with probability 1). On the other hand, if $A_j = A_i$ but $m_j \neq m_i$, then $c_i \neq c_j$ with probability 1. Otherwise, the distribution of ciphertext c_i is independent of the distribution of any c_j that is encrypted with different parameters $A_j \neq A_i$ or $|m_j| \neq |m_i|$ (by definition of **URI**).

In system $\mathsf{sim}^{\mathtt{E}}\mathbf{RSEC}_{\mathrm{TLS}}$, the simulator gets the pair (ℓ, k), where $\ell = |m_i|$ and k is the number of the first input that equals (A_i, m_i). This allows sim to consistently reflect repetitions of ciphertexts. If $k = i$, i.e., if the message is new, a uniformly random ciphertext c_i of length $\ell + \lambda$ is output and stored for future reference. By

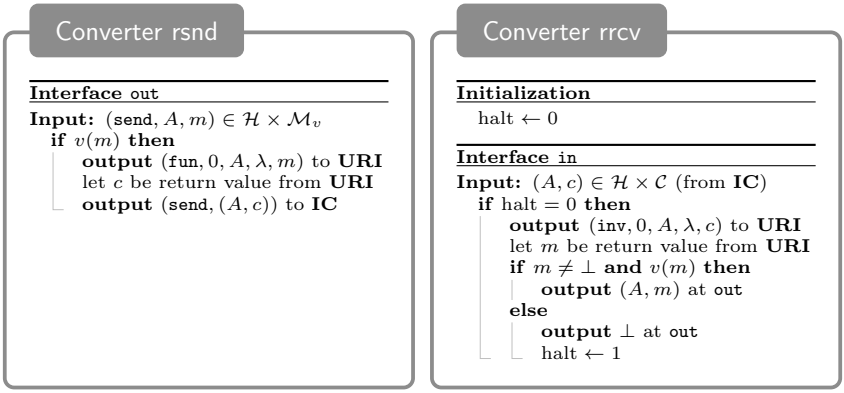


Figure 4.9: The converters for the sender (left) and the receiver (right).

comparing the behavior to $\mathsf{rsnd}^{\mathtt{A}}_{\lambda}\mathsf{rrcv}^{\mathtt{B}}_{\lambda}[\mathbf{URI}, \mathbf{IC}]$, we observe that the two systems have the same output distribution, as long as there is no collision to a previous ciphertext c_j that has been generated on input (A_j, m_j) with $A_j = A_i$ and $|m_i| = |m_j|$ but $m_i \neq m_j$. The probability of such a collision is bounded by

$$\frac{q \cdot (q-1)}{2} \cdot |\Sigma|^{-\lambda}. \tag{4.7}$$

On the ith input $(\mathtt{inject}, A_i, c_i)$ at interface $\mathtt{E}$: In the following, we assume that interface $\mathtt{B}$ has not output $\perp$ already, as otherwise the two systems behave identically anyway.

In $\mathsf{rsnd}^{\mathtt{A}}_{\lambda}\mathsf{rrcv}^{\mathtt{B}}_{\lambda}[\mathbf{URI}, \mathbf{IC}]$, rsnd queries $\mathbf{URI}$ with input $(\mathtt{inv}, 0, A_i, \lambda, c_i)$. In the following analysis, Let $\ell := |c_i| - \lambda$ and let denote $q_{\mathtt{A}}^{A_i,\ell}$ the total number of distinct queries $(\mathtt{send}, A_i, m)$ with $\ell = |m|$ at interface $\mathtt{A}$.

When rrcv decrypts ciphertext c_i of length $\ell+\lambda$ using associated data A_i, then there are two cases to consider: either $\mathbf{URI}$ has already generated a function mapping messages of length ℓ to ciphertexts of length $\ell + \lambda$ or $\mathbf{URI}$ has not yet generated such a function yet.

1. If $\mathbf{URI}$ has generated an injective function already, then there

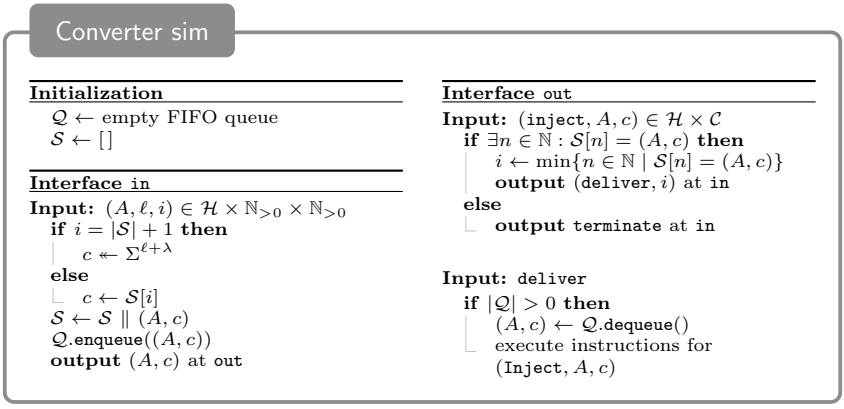


Figure 4.10: Simulator for the security condition of the construction of $\mathbf{RSEC}_{\mathrm{TLS}}$ from $\mathbf{RIC}_{\lambda}$.

have been $q_{\mathtt{A}}^{A_i,\ell}$ fixed message-ciphertext pairs (m'_j, c'_j) generated by **URI**. Clearly, if c_i equals one of the c'_j, then the output at interface B is m'_j. Otherwise, there is only an output at interface B if c_i has a preimage under the random injection and since there are $|\Sigma^\ell \cap \mathcal{M}_v| - q_{\mathtt{A}}^{A_i,\ell}$ possible valid preimages left and each one is assigned to one of $|\Sigma|^{\ell+\lambda} - q_{\mathtt{A}}^{A_i,\ell}$ ciphertexts, the probability that c_i has a preimage (and consequently that B produces an output) is at most

$$\frac{|\Sigma^\ell \cap \mathcal{M}_v| - q_{\mathtt{A}}^{A_i,\ell}}{|\Sigma|^{\ell+\lambda} - q_{\mathtt{A}}^{A_i,\ell}} \leq \frac{|\Sigma^\ell \cap \mathcal{M}_v|}{|\Sigma|^{\ell+\lambda}} \leq d_v \cdot |\Sigma|^{-\lambda}, \tag{4.8}$$

where we used $\frac{x-d}{y-d} \leq \frac{x}{y}$ for $x \leq y$ and $0 \leq d < y$ in the first step and the definition of d_v in the second step.

2. The second case follows trivially by observing that if **URI** has not generated a function yet, then $q_{\mathtt{A}}^{A_i,\ell} = 0$.

In $\mathsf{sim}^{\mathtt{E}}\mathbf{RSEC}_{\mathrm{TLS}}$, if the injected pair (A_i, c_i) is equal to a previously generated pair (E_j, c_j), the simulator outputs $(\texttt{deliver}, j)$ at the inner interface to instruct the channel to replay the jth message, which generates the output (A_j, m_j) at interface B.

On the other hand, if the pair (A_i, c_i) is new, sim terminates the channel. This element is guaranteed to decrypt to $\perp$ on any decryption attempt at interface $\mathtt{B}$.

System $\mathsf{sim}^{\mathtt{E}}\mathbf{RSEC}_{\mathrm{TLS}}$ reflects the behavior of $\mathsf{enc}_{\Pi}{}^{\mathtt{A}}\mathsf{dec}_{\Pi}{}^{\mathtt{B}}[\mathbf{URI}, \mathbf{IC}]$ as long as the converter rrcv_{λ} does not generate a valid output upon its first attempt to decrypt an injected associated data-ciphertext pair (A, c) that has not been generated in reaction to a send-query at interface $\mathtt{A}$. In such a case, sim would always insert an empty element and provoke the output $\perp$. However, the probability of rrcv_{λ} producing a valid output upon decryption of a new associated data-ciphertext pair is bounded in (4.8).

On the ith input $(\mathtt{deliver})$ **at interface** $\mathtt{E}$**:** In $\mathsf{rsnd}_{\lambda}^{\mathtt{A}}\mathsf{rrcv}_{\lambda}^{\mathtt{B}}[\mathbf{URI}, \mathbf{IC}]$, the front element, say (A, c) of the sender queue (within $\mathbf{IC}$) is output to rrcv that tries to decrypt the associated data-ciphertext pair. Furthermore, since the pair is a valid encryption, the message is output at interface $\mathtt{B}$.

In $\mathsf{sim}^{\mathtt{E}}\mathbf{RSEC}_{\mathrm{TLS}}$, the simulator retrieves the front element (A, c) of queue $\mathcal{Q}$ which simulates the sender queue of the real execution of the protocol. The simulator executes the same instructions as for $(\mathtt{inject}, A, c)$ which yields the same behavior for both systems, since also in this case, the pair (A, c) is in the simulator's list of valid associated data-ciphertext pairs.

This concludes the analysis and we can bound the distinguishing advantage by using inequalities (4.7) and (4.8). The analysis of the correctness condition (4.5) is straightforward and omitted. □

Chapter 5

Signcryption

5.1 Introduction

We conclude our study on cryptographic building blocks for secure communication with a critical look at signcryption and the corresponding basic security notions. Unlike the previous studies, we take a fresh approach at answering the question what signcryption schemes should ideally achieve. This does not only lead to a better understanding through a constructive definition — showing the idealization of protecting communication in a public-key setting — but it also allows us to see which game-based security definition should be considered the standard notion for signcryption.

5.1.1 Motivation

Signcryption is a public-key cryptographic primitive introduced by Zheng [Zhe97] in 1997, which simultaneously provides two fundamental cryptographic goals: *confidentiality* and *authenticity*. Intuitively, the first property ensures that no one except the intended recipient should be able to learn anything about a sent message, and this is typically achieved by means of an encryption algorithm, and the second property ensures that the receiver can verify that a message indeed originated from the claimed sender, which is typically achieved by employing a digital signature scheme. Signcryption can be seen as the public-key analogue of authenticated encryption and shares part of its motivation: by merging

the two security goals, one might gain practical efficiency and at the same time offer better usability to applications, since there is only a single scheme that needs to be employed. Since its introduction, several concrete schemes have emerged in the literature based on different hardness assumptions [Zhe97, ZI98, SZ00, LQ03, LQ04]. Also, new properties beyond the basic security goals have been introduced recently, such as identity-based [ML02, Boy03, LQ03, LBZ11, SSVPR10, SVVR12], hybrid [Den05], KEM-DEM-based [BD06], certificateless [BF08], verifiable [SSVPR10], attribute-based [PPB14, DDM15a], functional [DDM15b], or key invisible [WMAS13] signcryption schemes.

Identifying the right definition. Finding the initial, basic security definitions for signcryption proved to be a very subtle and challenging task. In fact, the original signcryption scheme by Zheng was formally proven secure only about ten years after its introduction by Baek, Steinfield, and Zheng [BSZ07]. While (symmetric) authenticated encryption was put on solid security definitions directly from the start, the basic security notions for signcryption have had a more difficult path and converged to a set of commonly agreed notions only recently [DZ10] and only thanks to the merits of a sequence of foundational works [An01, ADR02, BSZ07] that formally introduced what is now known as the *outsider security model* — the model that captures network attackers or an adversarial entity that registers public keys with a certificate authority — and the *insider security model* — the model that captures attacks of corrupted users, for example an a priori legitimate user whose private key got compromised.

Only little effort has subsequently been made to investigate what the security notions precisely mean and whether they provide the expected service to higher-level protocols. An initial approach to this question was taken in [GK07] where a functionality is presented that idealizes the process of using the signcryption algorithm to ensure unforgeability and confidentiality (focusing on the outsider security model) along the lines of the signature and public-key encryption functionality in the UC framework.

We significantly advance this line of research and provide a detailed application-centric analysis of the basic security notions of signcryption. We further believe that our analysis provides sufficient evidence to call insider security the standard notion for signcryption and we are able to

pinpoint which proposed variants of insider security are mainly relevant.

5.1.2 Specific Contributions

Defining an application scenario for signcryption. For the first time, we model a concrete and basic application scenario for signcryption schemes. For a public-key setting with potentially several parties, we make the following natural choice:

- An insecure network **Net**, where each user can register themselves with a unique identity and send and receive messages, and where a network attacker, say Eve, has full control over the network, including message delivery.
- A certificate authority **CA**, where users and the attacker Eve can register public keys in the name of the identity. The certificate authority only guarantees that there is exactly one value registered for an identity, but does not verify knowledge of, for example, a secret key.

While a key-compromise at one communication endpoint always implies security loss at the partner's endpoint in the shared-key setting, this need not be the case in the public-key setting where users are a-priori independent in principle. In order to model fine-grained attacks, we introduce an explicit memory resource **Mem** that models the storage of the secret values of each user. The storage is possibly compromised by an intruder, say Mallory, which models key compromise.

Defining the goal for signcryption. The security goal of signcryption can be identified in a very natural way: due to the nature of public-key cryptography, the security depends on which user gets compromised. In the best case, if a user is compromised, we have to give up just his respective security: this means that messages sent to this user can be read by the attacker, and the attacker can act in the name of this user. This directly gives rise to a notion of a secure network that gracefully degrades depending on which users gets compromised as described below. We denote this gracefully-degrading secure network by **SecNT** and its main properties are as follows:

1. If two uncompromised legitimate users communicate, then the secure network guarantees that the network attacker learns at most the length of the messages and the attacker cannot inject any message into this communication: the communication between them can be called secure.

2. If, however, the legitimate sender is compromised, but not the receiver, then the network allows the attacker to inject messages in the name of this sender. Still, Eve does not learn the contents of the messages to the receiver: the communication is thus only confidential.

3. If, on the other hand, the legitimate receiver is compromised, but not the sender, the secure network allows Eve to read the contents of the messages sent to this compromised user. Still, no messages can be injected into this communication: the communication is only authentic.

4. If both, sender and receiver, are compromised, then the network does not give any guarantee on their communication, Eve can read every message and inject anything at will.

The preferred insider security notion. Our approach can be used to formally identify those game-based notions that provable imply the above construction. For example, the security games in Figure 5.3 and Figure 5.2 are thus an adequate choice to model game-based insider security. Any choice can be called suitable as long as the chosen set of games implies the construction. The notions we use are in particular implied by what is denoted in [DZ10] as "multi-user insider confidentiality in the FSO/FUO-IND-CCA2 sense" and "multi-user insider unforgeable in the FSO/FUO-sUF-CMA sense", respectively. The presented games are, however, weaker in some respect as we outline in section 5.2. In general, finding weaker versions of the above mentioned insider security games that still imply the construction statement might be a helpful step in finding more efficient schemes that achieve the construction. In this vein, an interesting open problem is which game-based notion would actually be equivalent to our constructive formulation.

Graceful degradation thanks to insider security. One crucial point of our main result is that it is insider security that provably assures that the secure network degrades gracefully as a function of compromised keys and does not lose the security guarantees in a coarse-grained fashion (for example per pair of parties instead of a single party). This novel view underlines the importance of insider security as a distinctive feature that indeed assigns signcryption a special significance in actual deployments of network protocols. We note that its importance has been (and still is) overlooked by a substantial fraction of works. In particular, we contrast the line of previous works that propose, analyze and revisit signcryption schemes and their security, including [GK07, TP14, BSZ07], recent developments in practical lattice-based schemes [GM18], and one of the main references on the basic notions [DZ10, pages 29 and 46], that assign only little credit to the relevance of insider security.

5.1.3 The Constructive Cryptography Setting

In this chapter, we consider the general setting of constructive cryptography that allows to capture honest parties, a network attacker, and intruders (that model key compromise). The resources have interface sets $\mathcal{I} = \{\mathtt{P}_1, \ldots, \mathtt{P}_n, \mathtt{M}_1, \ldots, \mathtt{M}_n, \mathtt{E}\}$, where interface $\mathtt{P}_i$ can be thought of as being the access point of the ith honest party to the system. Interface $\mathtt{M}_i$ is the access point of an intruder (i.e., a hypothetical attacker entity like Mallory), and $\mathtt{E}$ is the access point of the network attacker Eve (also a hypothetical entity). We will recap the most important concepts from Section 2.3, such as filtered resources, where we think this is needed to better follow the results.

5.2 Signcryption: Game-Based Notions

We present the definition of Signcryption schemes and the relevant security games from [BSZ07].[1]

[1] We make two simplifications: first, as usual, we do not make domain parameter generation for the algorithms explicit, and second we simply call the key space $\mathcal{K}$ and do not introduce separate domains for private and public keys.

Definition 5.2.1. A signcryption scheme $\Psi = (\mathsf{Gen}_S, \mathsf{Gen}_R, \mathsf{Signcrypt}, \mathsf{Unsigncrypt})$ for *key space* $\mathcal{K}$, *message space* $\mathcal{M}$, and *signcryptext space* $\mathcal{S}$, is a collection of four (efficient) algorithms:

- A *sender key generation algorithm* Gen_S, which outputs a *sender key-pair* (sk_S, pk_S), i.e., the *sender private key* $sk_S \in \mathcal{K}$ and the *sender public key* $pk_S \in \mathcal{K}$, respectively. We write $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S$.
- A *receiver key generation algorithm* Gen_R, which outputs a *receiver key-pair* (sk_R, pk_R), i.e., the *receiver private key* $sk_R \in \mathcal{K}$ and the *receiver public key* $pk_R \in \mathcal{K}$, respectively. We write $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R$.
- A (possibly randomized) *signcryption algorithm* $\mathsf{Signcrypt}$, which takes as input a sender private key sk_S, a receiver public key pk_R, and a *message* $m \in \mathcal{M}$, and outputs a *signcryptext* $s \in \mathcal{S}$. We write $c \leftarrow \mathsf{Signcrypt}(sk_S, pk_R, m)$.
- A (usually deterministic) *unsigncryption algorithm* $\mathsf{Unsigncrypt}$, which takes as input a receiver private key sk_R, a sender public key pk_S, and a *signcryptext ("the ciphertext")* $s \in \mathcal{S}$, and outputs a *message* $m \in \mathcal{M}$, or a special symbol $\perp$. We write $m \leftarrow \mathsf{Unsigncrypt}(sk_R, pk_S, s)$.

The scheme is *correct* if for all sender key pairs (sk_S, pk_S) in the support of Gen_S, and for all receiver key pairs (sk_R, pk_R) in the support of Gen_R, and for all $m \in \mathcal{M}$ it holds that

$$\mathsf{Unsigncrypt}(sk_R, pk_S, (\mathsf{Signcrypt}(sk_S, pk_R, m)) = m.$$

5.2.1 Multi-User Outsider Security

As explained before, the multi-user outsider security model (MOS) has been introduced to capture network attackers trying to break the secure communication between a sender and a receiver. Two legitimate sender and receiver key-pairs are generated by the game and given to the adversary. Similar to authenticated encryption, the oracles of the games in Figure 5.1 allow the adversary to obtain signcryptexts of arbitrary messages (i.e., an encryption oracle) generated by the sender for a adversarially chosen receiver public key, and the messages corresponding to

chosen signcryptexts (i.e., a decryption oracle) when decrypted by the receiver w.r.t. an adversarially chosen sender public key. As for authenticated encryption, the goal of the adversary is to distinguish the scheme from an idealized version, i.e., an ideal game that only decrypts random messages on behalf of the legitimate sender (perfect confidentiality), and never allows a successful decryption w.r.t. to the legitimate sender public key (perfect integrity).

For clarity, we apply the oracle-silencing technique to capture the set of valid adversaries here: an adversary is not allowed to ask unsigncryption-queries for answers obtained from the signcryption oracle and w.r.t. the sender public-key. This is why upon such an invalid query, the adversary obtains the special symbol $\perp$. We note that the formalization used in this work, i.e., the so-called all-in-one definition (capturing both integrity and confidentiality) is introduced in [BBM18] and — just like the all-in-one formulation for authenticated encryption introduced in [RS06] — is simply equivalent to a definition where both notions are captured by separate games. For a proof we refer to [BBM18].

Definition 5.2.2. Let $\Psi = (\mathsf{Gen}_S, \mathsf{Gen}_R, \mathsf{Signcrypt}, \mathsf{Unsigncrypt})$ be a signcryption scheme and $\mathcal{A}$ a probabilistic algorithm. Consider games $\mathbf{Real}_{\Psi}^{\mathsf{MOS}}$ and $\mathbf{Ideal}_{\Psi}^{\mathsf{MOS}}$ from Figure 5.1. We define the *real-or-random multi-user outsider security advantage* of $\mathcal{A}$ as

$$\mathsf{Adv}_{\Psi,\mathcal{A}}^{\mathsf{MOS}} \coloneqq \Pr\left[\mathcal{A}^{\mathbf{Real}_{\Psi}^{\mathsf{MOS}}} = 1\right] - \Pr\left[\mathcal{A}^{\mathbf{Ideal}_{\Psi}^{\mathsf{MOS}}} = 1\right].$$

5.2.2 Multi-User Insider Security

The multi-user insider security model (MIS) captures two basic requirements, namely that encryptions should look random even if the sender-private key is known (but not the receiver private key) (MIS-Conf), and, no forgeries should be possible even if the receiver private key is known (but not the sender private key) (MIS-Auth). For both security goals, we state the traditional definitions and we introduce a weaker version thereof. As it turns out, for the basic application of signcryption schemes, the weaker version is sufficient.

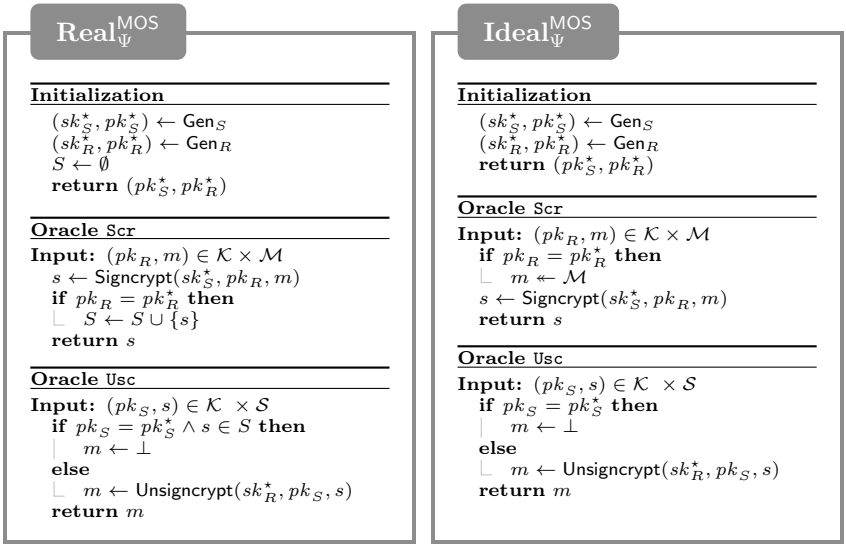


Figure 5.1: The games $\mathbf{Real}_\Psi^{\mathsf{MOS}}$ and $\mathbf{Ideal}_\Psi^{\mathsf{MOS}}$.

Confidentiality

The games capturing MIS-Conf are given in Figure 5.3 and are basically a CCA-style definition (using the real-or-random version).

We specify two variants of different strengths: the games that include the `Gen` oracle and the boxed statements constitute the weaker version which we use in this work. Intuitively, the weaker game does not allow the adversary to choose the randomness to generate keys. However, in both variants whenever the adversary makes an oracle call, he has to provide a *valid* key-pair. As commonly known, enforcing this is actually indispensable in order to avoid trivial attacks. For example, an attacker could specify a pair $(sk_S, 0)$ in a signcryption query, which allows him to unsigncrypt the respective result using the actual (correct) public key pk_S. We now state the formal definition:

Definition 5.2.3. Let $\Psi = (\mathsf{Gen}_S, \mathsf{Gen}_R, \mathsf{Signcrypt}, \mathsf{Unsigncrypt})$ be a signcryption scheme and $\mathcal{A}$ a probabilistic algorithm. We define the advantage of $\mathcal{A}$ in distinguishing $\mathbf{Real}_\Psi^{\mathsf{MIS\text{-}Conf}}$ and $\mathbf{Ideal}_\Psi^{\mathsf{MIS\text{-}Conf}}$ from

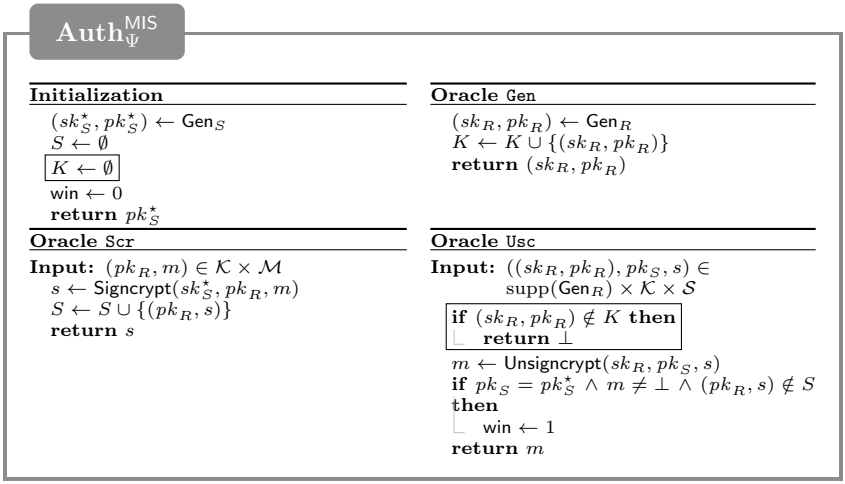


Figure 5.2: The forgery game $\mathbf{Auth}_\Psi^{\mathsf{MIS}}$. The game that includes the boxed statements (and the oracle **Gen**) constitutes the weaker version.

Figure 5.3 as

$$\mathsf{Adv}^{\mathsf{MIS\text{-}Conf}}_{\Psi,\mathcal{A}} := \Pr\left[\mathcal{A}^{\mathbf{Real}^{\mathsf{MIS\text{-}Conf}}_\Psi} = 1\right] - \Pr\left[\mathcal{A}^{\mathbf{Ideal}^{\mathsf{MIS\text{-}Conf}}_\Psi} = 1\right].$$

We consider the weaker game including the boxed lines. The version which excludes those lines, and also the `Gen` oracle, would yield the definition traditionally found in the literature.

Authenticity

The forgery game $\mathbf{Auth}_\Psi^{\mathsf{MIS}}$ is given in Figure 5.2. We again give two variants as for confidentiality before. We directly state the relevant definition:

Definition 5.2.4. Let $\Psi = (\mathsf{Gen}_S, \mathsf{Gen}_R, \mathsf{Signcrypt}, \mathsf{Unsigncrypt})$ be a signcryption scheme and $\mathcal{A}$ a probabilistic algorithm. We define the advantage of $\mathcal{A}$ when interacting with $\mathbf{Auth}_\Psi^{\mathsf{MIS}}$ from Figure 5.2 as

$$\mathsf{Adv}^{\mathsf{MIS\text{-}Auth}}_{\Psi,\mathcal{A}} := \Pr\left[\mathcal{A}^{\mathbf{Auth}_\Psi^{\mathsf{MIS}}} \text{ sets } \mathsf{win}\right].$$

We consider the weaker game including the boxed lines. The version which excludes those lines, and also the `Gen` oracle, would yield the definition traditionally found in the literature.

$\mathbf{Real}_{\Psi}^{\text{MIS-Conf}}$

Initialization

$(sk_R^\star, pk_R^\star) \leftarrow \mathsf{Gen}_R$
$S \leftarrow \emptyset$
$\boxed{K \leftarrow \emptyset}$
return $pk_R^\star$

Oracle `Gen`

$(sk_S, pk_S) \leftarrow \mathsf{Gen}_S$
$K \leftarrow K \cup \{(sk_S, pk_S)\}$
return (sk_S, pk_S)

Oracle `Scr`

Input: $((sk_S, pk_S), pk_R, m) \in \mathrm{supp}(\mathsf{Gen}_S) \times \mathcal{K} \times \mathcal{M}$
$\boxed{\textbf{if } (sk_S, pk_S) \notin K \textbf{ then}}$
$\quad\boxed{\textbf{return } \bot}$
$s \leftarrow \mathsf{Signcrypt}(sk_S, pk_R, m)$
if $pk_R = pk_R^\star$ **then**
$\quad S \leftarrow S \cup \{(pk_S, s)\}$
return s

Oracle `Usc`

Input: $(pk_S, s) \in \mathcal{K} \times \mathcal{S}$
if $(pk_S, s) \in S$ **then**
$\quad m \leftarrow \bot$
else
$\quad m \leftarrow \mathsf{Unsigncrypt}(sk_R^\star, pk_S, s)$
return m

$\mathbf{Ideal}_{\Psi}^{\text{MIS-Conf}}$

Initialization

$(sk_R^\star, pk_R^\star) \leftarrow \mathsf{Gen}_R$
$S \leftarrow \emptyset$
$\boxed{K \leftarrow \emptyset}$
return $pk_R^\star$

Oracle `Gen`

$(sk_S, pk_S) \leftarrow \mathsf{Gen}_S$
$K \leftarrow K \cup \{(sk_S, pk_S)\}$
return (sk_S, pk_S)

Oracle `Scr`

Input: $((sk_S, pk_S), pk_R, m) \in \mathrm{supp}(\mathsf{Gen}_S) \times \mathcal{K} \times \mathcal{M}$
$\boxed{\textbf{if } (sk_S, pk_S) \notin K \textbf{ then}}$
$\quad\boxed{\textbf{return } \bot}$
if $pk_R = pk_R^\star$ **then**
$\quad m \twoheadleftarrow \mathcal{M}$
$s \leftarrow \mathsf{Signcrypt}(sk_S, pk_R, m)$
if $pk_R = pk_R^\star$ **then**
$\quad S \leftarrow S \cup \{(pk_S, s)\}$
return s

Oracle `Usc`

Input: $(pk_S, s) \in \mathcal{K} \times \mathcal{S}$
if $(pk_S, s) \in S$ **then**
$\quad m \leftarrow \bot$
else
$\quad m \leftarrow \mathsf{Unsigncrypt}(sk_R^\star, pk_S, s)$
return m

Figure 5.3: The games $\mathbf{Real}_{\Psi}^{\text{MIS-Conf}}$ and $\mathbf{Ideal}_{\Psi}^{\text{MIS-Conf}}$. The variant with the boxed statements (and oracle **Gen**) constitutes the weaker notion.

5.3 Gracefully-Degrading Secure Networks

In this section, we first introduce the new notion of gracefully-degrading secure networks as outlined in Section 5.1.2. We then formally introduce the assumed resources and the protocol converters and proof that we can construct a gracefully degrading secure network based on the security guarantees provided by the definitions of Section 5.2.

5.3.1 Definition

The resource we want to achieve is a secure network that gracefully degrades and is specified in Figure 5.4. This ideal network is basically a secure network. To see this, imagine there was no interface $\mathtt{M}_i$: then parties can register to the network (using some unique identifier ID) and can send and receive messages. In addition, the adversary learns the length of the message (in addition to sender and receiver identities), and cannot inject messages. Note that this behavior is enforced in Figure 5.4 by adding the associated identity after successful registration at interface $\mathtt{P}_i$ to the special set S only if there was no input `reveal` at interface $\mathtt{M}_i$. Now observe that the condition under which the network attacker can inject a message for some party identity ID includes that $\mathsf{ID} \notin S$. In addition, the network attacker learns only the length of the messages whenever a message is sent to an identity $\mathsf{ID} \in S$. Thus, since all registered identities of honest parties are in S, communication between any two of them is secure. Now, the input `reveal` is potentially available at interface $\mathtt{M}_i$ (this models the fact that the party is compromised). Whenever this input happens, then the corresponding party identity is not included in S. This means that the network attacker at interface $\mathtt{E}$ can inject messages on behalf of the identity registered at interface $\mathtt{P}_i$ and obtains the content of any message sent to $\mathtt{P}_i$. We see that only the security of $\mathtt{P}_i$ is affected as declared in Section 5.1.2.

To complete this description, note that the secure network outputs shared randomness between the intruder of party $\mathtt{P}_i$ and the network attacker. This models that in the ideal world, shared randomness is potentially available to the parties. This is indeed the case, since the network attacker learns signcryptexts that are created with the secret key leaked at interface $\mathtt{M}_i$. On a technical level, shared randomness is needed to achieve a consistent simulation. Note that in constructive cryptography,

we have local simulators (cf. Section 2.3) for each of the interfaces with potential dishonest behavior.

On using filters. In this chapter, we use a convenient tool of constructive cryptography that we introduced in cf. Section 2.3. Note that an intruder, as well as a network attacker, are hypothetical entities and we want to model that these capabilities are only potentially available. For example, at interface $\mathtt{M}_i$, the capability to reveal is only potentially available but not guaranteed. This means that we actually have to consider a filtered resource $\mathbf{SecNT}_{n\,\phi^{\text{ideal}}}$ and specify the filter converters for the interfaces.

As detailed in Section 2.3, this allows us in particular to give security guarantees for each subset of dishonest behavior, and to eventually prove that security degrades gracefully as a function of the interfaces that are dishonest. Looking ahead, the potentially available capability to compromise a party corresponds to the potentially available input `reveal` in the ideal world. Figure 5.6 illustrates an example instantiation to clarify the concepts.

Hence, at interfaces $\mathtt{M}_i$, the filter is given by the converter $\mathsf{0}$ that blocks all interaction (no "leakage"). At interface $\mathtt{E}$, the filter is given by the converter dlv that implements reliable message transfer[2], and finally, the filter of honest interfaces is given by the identity converter $\mathsf{1}$.

Overall, we can thus define the filter by $\phi^{\text{ideal}} := (\mathsf{1}, \ldots, \mathsf{1}, \mathsf{0}, \ldots, \mathsf{0}, \mathsf{dlv})$ for interfaces $\mathtt{P}_1, \ldots, \mathtt{P}_n, \mathtt{M}_1, \ldots, \mathtt{M}_n, \mathtt{E}$ for the resource $\mathbf{SecNT}_n$.

5.3.2 Assumed Resources

We introduce standard resources to model our concrete setting.

Insecure network. We assume a network resource $\mathbf{Net}_n$ that accepts, at each interface $\mathtt{P}_i$, $i \in [n]$, a registration query that assigns an identifier to that interface. Any bitstring $\mathsf{ID} \in \{0,1\}^*$ is valid, and uniqueness is enforced (reflecting IP-addresses). Subsequently, messages can be sent at this interface in the name of that identifier, by indicating the message

[2]More formally, upon any $(\cdot, \mathsf{ID}_s, \mathsf{ID}_r)$ at its inner interface from the network (at interface $\mathtt{E}$), the converter dlv responds with $(\mathtt{inject}, \cdot, \mathsf{ID}_s, \mathsf{ID}_r)$ to the network resource (and does not give any output at its outer interface and it does not react on any other input)

Resource $\mathbf{SecNT}_n$

Initialization

$r_1||r_2||\ldots||r_n \leftarrow (\{0,1\}^{2\kappa})^n$
▷ Common randomness
$I_1, I_2 \leftarrow$ empty tables
▷ Mapping interfaces and identities
$J, S \leftarrow \emptyset$ ▷ Registered identities
$L \leftarrow [\,]$ ▷ Inputs of parties
for $i = 1$ to n **do**
  $\mathsf{setupCompleted}_i \leftarrow \mathsf{false}$ ▷ Init. flag

Interface $\mathtt{M}_i$

Input: `reveal`
  $I_c \leftarrow I_c \cup \{i\}$
  if $\mathsf{setupCompleted}_i$ **then**
    output r_i at $\mathtt{M}_i$

Interface `E`

Input: $(\mathtt{deliver}, j) \in \mathbb{N}$
  if $j \leq |L|$ **then**
    Parse $L[j]$ as $(m, \mathsf{ID}_s, \mathsf{ID}_r)$
    $\mathtt{P} \leftarrow I_2[\mathsf{ID}_r]$
    output (m, ID_s) at `P`

Input: $(\mathtt{register}, \mathsf{ID}) \in \{0,1\}^*$
  if $\mathsf{ID} \notin J$ **then**
    $J \leftarrow J \cup \{\mathsf{ID}\}$
    output Success at `E`
  else
    output Fail at `E`

Input: $(\mathtt{inject}, m, \mathsf{ID}_s, \mathsf{ID}_r) \in \mathcal{M} \times \{0,1\}^* \times \{0,1\}^*$
  $\mathtt{P} \leftarrow I_2[\mathsf{ID}_r]$
  if $\mathsf{ID}_s \in J$ **and** $\mathtt{P} \neq \perp$ **and** $\mathsf{ID}_s \notin S$ **then**
    output (m, ID_s) at `P`

Input: `getCommonRand`
  output $\{(i, r_i) \mid i \in I_c\}$ at `E`

Input: `getMapping`
  output (I_1, I_2, J) at `E`

Interface $\mathtt{P}_i$

Input: $(\mathtt{register}, \mathsf{ID}) \in \{0,1\}^*$
  if $\mathsf{ID} \notin J \wedge I_1[\mathtt{P}_i] = \perp$ **then**
    $J \leftarrow J \cup \{\mathsf{ID}\}$
    $I_1[\mathtt{P}_i] \leftarrow \mathsf{ID}$
    $I_2[\mathsf{ID}] \leftarrow \mathtt{P}_i$
    $\mathsf{setupCompleted}_i \leftarrow \mathsf{true}$
    if $i \notin I_c$ **then**
      $S \leftarrow S \cup \{\mathsf{ID}\}$
    output Success at $\mathtt{P}_i$
  else
    output Fail at $\mathtt{P}_i$

Input: $(\mathtt{send}, m, \mathsf{ID}) \in \mathcal{M} \times \{0,1\}^*$
  $\mathsf{ID}_s \leftarrow I_1[\mathtt{P}_i]$
  if $\mathsf{ID}_s \neq \perp$ **and** $\mathsf{ID} \in J$ **then**
    if $\mathsf{ID} \in S$ **then**
      $L \leftarrow L||(m, \mathsf{ID}_s, \mathsf{ID})$
      output $(|m|, \mathsf{ID}_s, \mathsf{ID}_r)$ at `E`
    else
      output $(m, \mathsf{ID}_s, \mathsf{ID}_r)$ at `E`

Figure 5.4: The (unfiltered) behavior of the constructed resource.

content m and a destination identifier. Any request is leaked at interface $\mathtt{E}$ of the network (to the network attacker). Eve can further inject any message it wants to each destination address and indicate any source address as sender. At interface $\mathtt{E}$, these capabilities are *only potentially* available and thus not guaranteed. We thus specify a filter converter for this interface as before.

Memory. We model the local memory of each honest party by a memory resource $\mathbf{Mem}_n$. The memory can be thought of as being composed of n local memory modules. For the ease of exposition, we summarize these modules in one memory functionality that mimics this behavior (each party can read and write to *its* (and only this) memory location). The memory allows each party to store a value. In the construction, this will be the key storage. We make the storage explicit to model key compromises. To this end, we associate an intruder interface $\mathtt{M}_i$ to each party interface $\mathtt{P}_i$. At interface $\mathtt{M}_i$, the key is *only potentially* available to an intruder Mallory and thus not guaranteed. This means that we consider a filtered memory as an assumed resource where the filter only shields the capability at interfaces $\mathtt{M}_i$. Therefore, key-compromise attacks (or key leakage) is captured by removing the filter which allows us to model each key compromise as a separate event.

Certificate authority. The resource $\mathbf{CA}_n$ models a key registration functionality, and we denote it by certificate authority to stick to the common term in public-key infrastructures. The resource allows to register at an interface with an identity-value pair. The resource stores this assignment and does not accept any further registration with the same identity. The certificate authority is weak in the sense that it does not perform any further test and corresponds to typical formalizations of key registration functionalities. Any party can query to $(\mathtt{fetch}, \mathsf{ID})$ to retrieve the value registered for identity ID. Eve can register any value with any identity, under the constraint that the identity is not already registered. The capabilities at interface $\mathtt{E}$ are again not guaranteed and will be filtered as in the case of the network.

Summary. Overall, the assumed resource is the parallel composition of the resources introduced above, i.e., $[\mathbf{Net}_n, \mathbf{CA}_n, \mathbf{Mem}_n]_{\phi^{\text{real}}}$, where

ϕ^{real} is the filter that shields the memory (interfaces $\mathtt{M}_i$), the network, and the certificate authority (interface $\mathtt{E}$), and is exactly equal to the filter ϕ^{ideal} given in Section 5.3.1. The description using pseudo-code of all resources appears in Figure 5.5.

5.3.3 Construction

In the final section of this chapter, we show that a gracefully degrading secure network can be constructed based on the MOS-security and MIS-Conf and MIS-Auth of signcryption security.

Protocol

Recall from Section 2.3 that a protocol in constructive cryptography is formalized by a tuple of converters, where each converter is to be attached at its respective interface. We introduce a signcryption converter scr_Ψ, that is defined for a signcryption scheme $\Psi = (\mathsf{Gen}_S, \mathsf{Gen}_R, \mathsf{Signcrypt}, \mathsf{Unsigncrypt})$, and which will be attached at the honest interfaces. The converter specifies the actions that each party takes to secure the communication over the insecure network at interface $\mathtt{P}_i$. Upon a registration query, a party generates the two key-pairs required by the signcryption scheme, i.e., a sender key pair and a receiver key pair that it uses to send and receive message, respectively. It then tries to register its identity at the insecure network and tries to register the identity and the two public keys with the certificate authority. If everything succeeded, the converter stores the keys to its local memory. Otherwise, the initialization is not complete and the party remains un-initialized.

Upon sending a message, an initialized party retrieves the receiver public key of its intended communication partner, and signcrypts the message according to the signcryption scheme (and retrieves the secret key from the memory) and sends the signcryptext over the network (indicating the destination address). Upon receiving a pair (s, ID) consisting of a signcryptext and a candidate source address from the insecure network, it tries to unsigncrypt the given value and outputs the resulting message. The formal specification of this protocol converter appears in Figure 5.7.

The default behavior for potentially dishonest interfaces. Also the interfaces with potentially dishonest behavior formally need a converter.

Resource $\mathbf{CA}_n$

Initialization

$T \leftarrow$ empty table
$J \leftarrow \emptyset$

Interface $\mathtt{P}_i, i \in [n]$

Input: $(\mathtt{register}, \mathsf{ID}, val) \in \{0,1\}^* \times \{0,1\}^*$
  if $\mathsf{ID} \notin J$ **then**
    $J \leftarrow J \cup \{\mathsf{ID}\}$
    $T[\mathsf{ID}] \leftarrow val$
    output Success at $\mathtt{P}_i$
  else
    output Fail at $\mathtt{P}_i$

Input: $(\mathtt{fetch}, \mathsf{ID}) \in \{0,1\}^*$
  if $\mathsf{ID} \in J$ **then**
    output $T[\mathsf{ID}]$ at $\mathtt{P}_i$
  else
    output $\perp$ at $\mathtt{P}_i$

Interface E

Input: $(\mathtt{register}, \mathsf{ID}, val) \in \{0,1\}^* \times \{0,1\}^*$
  if $\mathsf{ID} \notin J$ **then**
    $T[\mathsf{ID}] \leftarrow val$
    $J \leftarrow J \cup \{\mathsf{ID}\}$
    output Success at E
  else
    output Fail at E

Input: fetchAll
  output (J, T) at E

Resource $\mathbf{Net}_n$

Initialization

$I_1, I_2 \leftarrow$ empty tables
$J \leftarrow \emptyset$

Interface $\mathtt{P}_i, i \in [n]$

Input: $(\mathtt{register}, \mathsf{ID}) \in \{0,1\}^*$
  if $\mathsf{ID} \notin J \ \wedge \ I_1[\mathtt{P}_i] = \perp$ **then**
    $J \leftarrow J \cup \{\mathsf{ID}\}$
    $I_1[\mathtt{P}_i] \leftarrow \mathsf{ID}$
    $I_2[\mathsf{ID}] \leftarrow \mathtt{P}_i$
    output Success at $\mathtt{P}_i$
  else
    output Fail at $\mathtt{P}_i$

Input: $(\mathtt{send}, m, \mathsf{ID}_r) \in \mathcal{M} \times \{0,1\}^*$
  $\mathsf{ID}_s \leftarrow I_1[\mathtt{P}_i]$
  if $\mathsf{ID}_s \neq \perp$ **then**
    output $(m, \mathsf{ID}_s, \mathsf{ID}_r)$ at E

Interface E

Input: $(\mathtt{inject}, m, \mathsf{ID}_s, \mathsf{ID}_r) \in \mathcal{M} \times \{0,1\}^* \times \{0,1\}^*$
  if $\mathsf{ID}_r \in J$ **then**
    $\mathtt{P} \leftarrow I_2[\mathsf{ID}_r]$
    output (m, ID_s) at P

Resource $\mathbf{Mem}_n$

Initialization

$val \leftarrow \perp$

Interface $\mathtt{P}_i, i \in [n]$

Input: $(\mathtt{store}, x) \in \{0,1\}^*$
  $val \leftarrow x$

Input: read
  output val at $\mathtt{P}_i$

Interface $\mathtt{M}_i, i \in [n]$

Input: reveal
  output val at $\mathtt{M}_i$

Figure 5.5: The (unfiltered) behavior of the assumed resources.

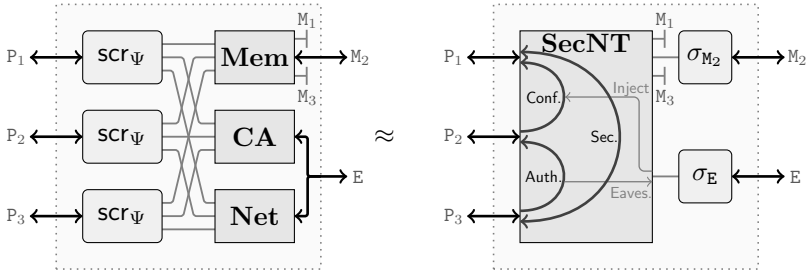


Figure 5.6: Illustration of the construction notion. Left (real world): Three parties running the protocol and where the second party's key got compromised. Right (ideal world): The secure network resource (with simulators) that guarantees secure communication between $\mathtt{P}_1$ and $\mathtt{P}_3$, but for example only confidential communication from party $\mathtt{P}_2$ to party $\mathtt{P}_1$, and only authentic communication from party $\mathtt{P}_3$ to party $\mathtt{P}_2$.

These are, however, quite simple: the intruder is assumed to perform no additional operation (the filter is not removed and exports no capability) and this converter is therefore simply the identity converter $\mathsf{1}$. The same holds for the network attacker where no additional operation needs to be specified.

Summary. The protocol for the assumed resource (with interface set $\{\mathtt{P}_1, \ldots, \mathtt{P}_n, \mathtt{M}_1, \ldots, \mathtt{M}_n, \mathtt{E}\}$) defined above can be succinctly defined by $\pi^{\Psi} := (\mathsf{scr}_{\Psi}, \ldots, \mathsf{scr}_{\Psi}, \mathsf{1}, \ldots, \mathsf{1}, \mathsf{1})$. It remains to prove that this protocol achieves the intended construction.

Security Proof

We are now ready to formally state the main theorem of this chapter.

Theorem 5.3.1. *Let Ψ be a signcryption scheme, let $n > 0$ be an integer, and let κ be an upper bound on the randomness used in one invocation of the key-generation algorithm. Then, the associated protocol $\pi^{\Psi} := (\mathsf{scr}_{\Psi}, \ldots, \mathsf{scr}_{\Psi}, \mathsf{1}, \ldots, \mathsf{1}, \mathsf{1})$ constructs the gracefully-degrading secure network from an insecure network, a certificate authority, and a memory resource within $\varepsilon(\cdot)$ and with respect to potentially dishonest*

Converter scr_{Ψ}

Initialization

$\mathsf{ID}_{\text{reg}} \leftarrow \perp$

Interface `out`

Input: $(\texttt{register}, \mathsf{ID}) \in \{0,1\}^*$
- **if** $\mathsf{ID}_{\text{reg}} = \perp$ **then**
  - $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S$
  - $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R$
  - $val \leftarrow [sk_S, pk_S, sk_R, pk_R]$
  - **output** $(\texttt{register}, \mathsf{ID}, (pk_S, pk_R))$ at `in` to **CA**
  - Let ret_{ca} be the return value
  - **if** $\mathsf{ret}_{\text{ca}} = \mathsf{Success}$ **then**
    - **output** $(\texttt{register}, \mathsf{ID})$ at `in` to **Net**
    - Let $\mathsf{ret}_{\text{net}}$ be the return value
    - **if** $\mathsf{ret}_{\text{net}} = \mathsf{Success}$ **then**
      - $\mathsf{ID}_{\text{reg}} \leftarrow \mathsf{ID}$
      - **output** $(\texttt{store}, val)$ at `in` to **Mem**
      - **output** Success at `out`
    - **else**
      - **output** Fail at `out`
  - **else**
    - **output** Fail at `out`

Input: $(\texttt{send}, m, \mathsf{ID}) \in \mathcal{M} \times \{0,1\}^*$
- **if** $\mathsf{ID}_{\text{reg}} \neq \perp$ **then**
  - **output** `read` at `in` to **Mem**
  - Let $val = [sk_S, pk_S, sk_R, pk_R]$ be the return value
  - **output** $(\texttt{fetch}, \mathsf{ID})$ at `in` to **CA**
  - Let ret_{ca} be the return value
  - Parse ret_{ca} as pair (pk'_S, pk'_R)
  - **if** $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**
    - $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m)$
    - **output** $(\texttt{send}, s, \mathsf{ID})$ at `in` to **Net**

Interface `in`

Input: $(s, \mathsf{ID}) \in \mathcal{S} \times \{0,1\}^*$ from **Net**
- **if** $\mathsf{ID}_{\text{reg}} \neq \perp$ **then**
  - **output** `read` at `in` to **Mem**
  - Let $val = [sk_S, pk_S, sk_R, pk_R]$ be the return value
  - **output** $(\texttt{fetch}, \mathsf{ID})$ at `in` to **CA**
  - Let ret_{ca} be the return value
  - Parse ret_{ca} as pair (pk'_S, pk'_R)
  - **if** $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**
    - $m \leftarrow \mathsf{Unsigncrypt}(sk_R, pk'_S, s)$
    - **if** $m \neq \perp$ **then**
      - **output** (m, ID) at `out`

Figure 5.7: The signcryption converter.

$\mathcal{U} := \{\mathtt{M}_1, \ldots, \mathtt{M}_n, \mathtt{E}\}$, *i.e.,*

$$[\mathbf{Net}_n, \mathbf{CA}_n, \mathbf{Mem}_n]_{\phi^{\text{real}}} \overset{(\pi^{\Psi}, \varepsilon, \mathcal{U})}{\Longmapsto} \mathbf{SecNT}_{n\,\phi^{\text{ideal}}},$$

for $\varepsilon(\mathbf{D}) := n^2 \cdot \mathsf{Adv}^{\mathsf{MOS}}_{\Psi,\rho_1(\mathbf{D})} + n \cdot \mathsf{Adv}^{\mathsf{MIS\text{-}Auth}}_{\Psi,\rho_2(\mathbf{D})} + n \cdot \mathsf{Adv}^{\mathsf{MIS\text{-}Conf}}_{\Psi,\rho_3(\mathbf{D})}$*, where the black-box reductions* ρ_1*,* ρ_2*, and* ρ_3 *map a distinguisher to an adversary (of essentially the same efficiency) and are defined below in the proofs for Claim 1, Claim 2, and Claim 3, respectively.*

Stated differently, if the signcryption scheme is secure in the respective multi-user, outsider-security and insider-security model, then the construction is achieved.

Proof. To prove Theorem 5.3.1, we specify the two converters σ_{net} and σ_{mem} in Figures 5.8 and 5.9 and prove that equation (2.2) is fulfilled for $\sigma = (\sigma_{\mathtt{M}_1}, \ldots, \sigma_{\mathtt{M}_n}, \sigma_{\mathtt{E}})$, where $\sigma_{\mathtt{E}} := \sigma_{\text{net}}$ and $\sigma_{\mathtt{M}_i} := \sigma_{\text{mem}}$ and for the above choice of $\varepsilon(.)$. In particular, we show that for any subset $\mathcal{C} \subseteq \mathcal{U}$ we have that

$$\Delta^{\mathbf{D}}(\pi^{\Psi}_{\mathcal{C}}\, \phi^{\text{real}}_{\mathcal{C}}[\mathbf{Net}_n, \mathbf{CA}_n, \mathbf{Mem}_n],\, \sigma_{\mathcal{C}}\, \phi^{\text{ideal}}_{\mathcal{C}}\mathbf{SecNT}_n) \leq \varepsilon(\mathbf{D}), \quad (5.1)$$

for any distinguisher $\mathbf{D}$. Fix any set $\mathcal{C} \subseteq \{\mathtt{M}_i, \ldots, \mathtt{M}_n, \mathtt{E}\}$. We first observe that if $\mathtt{E} \notin \mathcal{C}$, then the real and the ideal world are perfectly indistinguishable (i.e., the distinguishing advantage is zero for any distinguisher): both systems behave like a secure network, and for any interface $\mathtt{M}_i \in \mathcal{C}$, two signcryption key pairs are leaked. Since the network attacker is not present, this has no observable effect to the security properties, and since the signcryption scheme is assumed to be correct, every messages sent can be correctly received.

Without loss of generality we hence assume that $\mathtt{E} \in \mathcal{C}$. The set $\mathcal{C}$ induces a special corruption set $\mathcal{Z} \subseteq \{\mathtt{M}_1, \ldots, \mathtt{M}_n\}$, i.e., $\mathtt{M}_i \in \mathcal{Z} \leftrightarrow \mathtt{M}_i \in \mathcal{C}$. The set $\mathcal{Z}$ intuitively describes the corruption set, i.e., the set of parties whose keys are stolen.

Overview. We prove the statement by a "game-hopping" argument: We start with the real world $\mathbf{H}^{\mathcal{Z}}_0$ which is equivalent to the real world $\pi^{\Psi}_{\mathcal{C}}\, \phi^{\text{real}}_{\mathcal{C}}[\mathbf{Net}_n, \mathbf{CA}_n, \mathbf{Mem}_n]$ with $\mathcal{C} := \{\mathtt{E}\} \cup \mathcal{Z}$, and end with system $\mathbf{H}^{\mathcal{Z}}_6$ which is equivalent to ideal world $\sigma_{\mathcal{C}}\, \phi^{\text{ideal}}_{\mathcal{C}}\mathbf{SecNT}_n$ with $\mathcal{C} := \{\mathtt{E}\} \cup \mathcal{Z}$.

Converter σ_{net}

Initialization

$T \leftarrow$ empty table; $val_{\mathtt{P}_k} \leftarrow \perp$ for all $k \in [n]$; $j \leftarrow 0$

Interface `out`

Input: $(\mathtt{inject}, s, \mathsf{ID}_s, \mathsf{ID}_r)$ (for **Net**)
UpdateKeyTable ▷ $s \in \mathcal{S}$
Let $val_{\mathtt{P}_i} = [sk_S, pk_S, sk_R, pk_R]$
Parse $T[\mathsf{ID}_s]$ as (pk'_S, pk'_R)
if $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**
  if $\exists j : L_{sim}[c, \mathsf{ID}_s, \mathsf{ID}_r] = j\}$ **then**
    $j \leftarrow L_{sim}[c, \mathsf{ID}_s, \mathsf{ID}_r]$
    output $(\mathtt{deliver}, j)$ to **SecNT**
  else
    $m \leftarrow \mathsf{Unsigncrypt}(sk_R, pk'_S, s)$
    if $m \neq \perp$ **then**
      $(\mathtt{inject}, m, \mathsf{ID}_s, \mathsf{ID}_r)$

Input: $(\mathtt{register}, \mathsf{ID}, val)$ (for **CA**)
output $(\mathtt{register}, \mathsf{ID})$ at **SecNT**
Let v be the returned answer
if $v = \mathsf{Success}$ **then**
  $T[\mathsf{ID}] \leftarrow val$
  output Success at `E`
else
  output Fail at `E`

Input: `fetchAll` (for **CA**)
UpdateKeyTable
output `getMapping` at `in` to **SecNT**
Let (I_1, I_2, J) be the returned value
output (J, T) at `out`

Interface `in`

Input: $(v, \mathsf{ID}_s, \mathsf{ID}_r)$ (from **Net**)
UpdateKeyTable ▷ $v \in \mathcal{M}$ or $v \in \mathbb{N}$
output `getMapping` at `in` to **SecNT**
Let (I_1, I_2, J) be the returned value
Let $\mathtt{P}_i \leftarrow I_2[\mathsf{ID}_s]$
Parse val_i as $[sk_S, pk_S, sk_R, pk_R]$
Parse $T[\mathsf{ID}]$ as (pk'_S, pk'_R)
if $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**
  if $v \in \mathbb{N}$ **then**
    Let $\ell \leftarrow v$
    $m^* \twoheadleftarrow \{0,1\}^\ell$
    $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
  else
    $m^* \leftarrow v$
    $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
  $L_{sim}[c, \mathsf{ID}_s, \mathsf{ID}_r] \leftarrow j$
  $j \leftarrow j + 1$
  output $(s, \mathsf{ID}_s, \mathsf{ID}_r)$ at `E`

Helper procedure of this simulator:
procedure UpdateKeyTable
  output `getMapping` at `in` to **SecNT**
  Let (I_1, I_2, J) be the returned value
  output `getCommonRand` to **SecNT**
  Let $Z := \{(i, r_i)\}$ be the return value
  for each $\mathsf{ID} \in J$ **do**
    if $I_2[\mathsf{ID}] \neq \perp \ \wedge\ T[\mathsf{ID}] = \perp$ **then**
      Let $\mathtt{P}_k \leftarrow I_2[\mathsf{ID}]$
      if $(k, \cdot) \notin Z$ **then** $(r_k^1, r_k^2) \twoheadleftarrow \{0,1\}^{2\kappa}$
      $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S(r_k^1)$
      $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R(r_k^2)$
      $T[\mathsf{ID}] \leftarrow (pk_S, pk_R)$
      $val_{\mathtt{P}_k} \leftarrow [sk_S, pk_S, sk_R, pk_R]$

Figure 5.8: The simulator converter for the network attacker at interface `E`.

Converter σ_{mem}

Initialization

$val \leftarrow \bot$
output `reveal` at `in` to **SecNT** ▷ Compromise this party in the ideal world.

Interface `out`

Input: `reveal`
  if $val = \bot$ **then**
    output `reveal` at `in` to **SecNT**
    Let r be the returned value
    if r equals $r^1||r^2 \in \{0,1\}^{2\kappa}$ **then**
      $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S(r^1)$
      $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R(r^2)$
      $val \leftarrow [sk_S, pk_S, sk_R, pk_R]$
  output val at `out`

Figure 5.9: The simulator converter for the intruder, for each interface $\mathtt{M}_i$.

Each hop in this sequence is justified by careful syntactic inspection of the differences of the systems and their difference is either 0 (in case of syntactic modifications that do not affect the behavior) or can be bounded by the respective advantage of an attacker against the security games of signcryption by means of a reduction. Note that we prove the statement for a insider-security notion that is implied by the traditional insider-security notion. By transitivity, Theorem 5.3.1 also holds with respect to the traditional notion.

Sequence of hybrid worlds. Let $\mathcal{Z}$ be an arbitrary and fixed corruption set. We describe the systems below informally, indicating what is the change from one hybrid to the next. The code is given later and includes graphical indications what changes from one hybrid to the next. In total, we do eight steps as follows:

1.) The first hybrid system $\mathbf{H}_0^{\mathcal{Z}}$, depicted in Figure 5.10, describes the behavior of the real world, where we plugged together the protocol converters and the assumed system. We further introduce some new variables such as bad_1 and bad_2 that do not have an impact on the behavior.

2.) The second hybrid system $\mathbf{H}_1^{\mathcal{Z}}$, first depicted in Figure 5.11, is a slight modification of the first, where we replace encryptions of messages by encryptions of random messages, in case the communication is between two honest parties whose keys were not stolen. Furthermore, the adversary cannot inject messages for any such pairs of parties. The variable bad_1 is true, if and only if the adversary succeeds in breaking the security between any such pair. Note that the set S computed by the hybrids comprises all parties which are not corrupted, that is, all parties not in $\mathcal{Z}$.

For the difference between the first and second hybrid, we can prove the following claim.

Claim 1. *For any distinguisher* $\mathbf{D}$ *we have* $\Delta^{\mathbf{D}}(\mathbf{H}_0^{\mathcal{Z}}, \mathbf{H}_1^{\mathcal{Z}}) \leq n^2 \cdot \mathsf{Adv}^{\mathsf{MOS}}_{\Psi,\rho_{\mathcal{Z}}(\mathbf{D})}$*, that is, a (successful) distinguisher* $\mathbf{D}$ *for* $\mathbf{H}_0^{\mathcal{Z}}$ *and* $\mathbf{H}_1^{\mathcal{Z}}$ *can be transformed into a (successful) distinguisher* $\mathcal{A} = \rho_{\mathcal{Z}}(\mathbf{D})$ *(defined in the proof) for* $\mathbf{Real}^{\mathsf{MOS}}_{\Psi}$ *and* $\mathbf{Ideal}^{\mathsf{MOS}}_{\Psi}$*.*

Proof of claim. The idea is to use a standard hybrid argument on the set of all pairs of users which (pairwise) do not leak their secret key in the real world, that is, the set of all pairs of users which in the ideal world can communicate in a secure (both confidential and authentic) fashion. More precisely, we select one of those pairs of users uniformly at random, and for their (mono-directional) communications and design a reduction $\rho_{\mathcal{Z}}(\mathbf{D})$ that uses the oracles provided to the adversary by the security experiment to emulate the correct view towards $\mathbf{D}$.

Let the system $\mathbf{D}$ be a distinguisher for $\mathbf{H}_0^{\mathcal{Z}}$ and $\mathbf{H}_1^{\mathcal{Z}}$, and for the set $\mathcal{H} \coloneqq \{i \in [n] \,|\, \mathtt{M}_i \notin \mathcal{Z}\}$ of indexes of uncorrupted parties, let $\mathcal{L} \coloneqq \mathcal{H} \times \mathcal{H}$ denote the set of all ℓ^2 two-element[3] tuples over the set $\mathcal{H}$, with $\ell \coloneqq |\mathcal{H}|$. Let also fix an order over $\mathcal{L}$, that is, fix some efficiently computable bijection $\omega : [\ell^2] \to \mathcal{L}$ as well as its efficiently computable inverse map $\omega^{-1} : \mathcal{L} \to [\ell^2]$. We construct an adversary $\mathcal{A}$ for distinguishing $\mathbf{Real}^{\mathsf{MOS}}_{\Psi}$ from $\mathbf{Ideal}^{\mathsf{MOS}}_{\Psi}$ using distinguisher $\mathbf{D}$ via a reduction $\rho_{\mathcal{Z}}(\cdot)$, denoted $\mathcal{A} \coloneqq \rho_{\mathcal{Z}}(\mathbf{D})$ that will depend on the corruption set $\mathcal{Z}$.

The reduction works by first choosing an index t uniformly at random from $[\ell^2]$, and then computing the pair of indexes of *designated parties* $(S, R) \coloneqq \omega(t)$ (thus $S, R \in [n]$), corresponding to the *designated sender*

[3]Note that same-element tuples, i.e. $(\mathsf{ID}, \mathsf{ID})$, for $\mathsf{ID} \in \mathcal{H}$, are also included in $\mathcal{L}$.

$\mathtt{P}_S$ and the *designated receiver* $\mathtt{P}_R$, respectively. In the following, let $\mathcal{A}_t$ be the same as $\mathcal{A}$ but where the index t is fixed instead of uniformly randomly selected. For a random variable T uniformly distributed over $[\ell^2]$, this implies $\mathcal{A} = \mathcal{A}_T$. Recall that when $\mathcal{A}_t$ interacts with $\mathbf{Real}_{\Psi}^{\mathsf{MOS}}$ or $\mathbf{Ideal}_{\Psi}^{\mathsf{MOS}}$, it receives a pair of public keys, a sender public key $pk_S^\star$, which will be set as $\mathtt{P}_S$'s sender public key, and receiver public key $pk_R^\star$, which will be set as $\mathtt{P}_R$'s receiver public key. Upon registration of an identity via an honest interface, $\mathcal{A}_t$ generates and stores both sender and receiver key-pairs for the respective user, except that for party $\mathtt{P}_S$ only a receiver key-pair is generated and stored and for party $\mathtt{P}_R$ only a sender key-pair is generated and stored (recall that for both those parties $\mathcal{A}_t$ uses one of the public keys provided by either $\mathbf{Real}_{\Psi}^{\mathsf{MOS}}$ or $\mathbf{Ideal}_{\Psi}^{\mathsf{MOS}}$, whereas the corresponding secret keys are "hard-coded" into the provided oracles). Upon registration directly at interface $\mathtt{E}$, $\mathcal{A}_t$ internally stores the mapping of this identity to the registered public key. Whenever `reveal` is input at interface $\mathtt{M}_i \in \mathcal{Z}$ (i.e., $i \notin \mathcal{H}$), $\mathcal{A}_t$ returns the two generated key-pairs to the distinguisher $\mathbf{D}$ (and $\perp$ if $\mathtt{M}_i \notin \mathcal{Z}$, i.e., $i \in \mathcal{H}$). Similarly, it is easy for $\mathcal{A}_t$ to generate the mapping of identities to public keys when asked to reveal this information via a query `fetchAll`. We now describe the behavior of $\mathcal{A}_t$ on the remaining inputs.

On input $(\mathtt{send}, m, \mathsf{ID})$ **at interface** $\mathtt{P}_i$**:** The reduction $\mathcal{A}_t$ retrieves $\mathsf{ID}_{\mathtt{P}_i}$ and $\mathtt{P}_j$ from ID (recall that $i, j \in [n]$), and if both parties have previously successfully registered, $\mathcal{A}_t$ performs the following case distinction:

- If $(i, j) \notin \mathcal{L}$, then the message m is signcrypted into s using $\mathtt{P}_i$'s sender private key and $\mathtt{P}_j$'s receiver public key, and $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ is output at $\mathtt{E}$.
- If $(i, j) \in \mathcal{L}$, then the further case distinction is made:
  - If $\omega^{-1}(i, j) < t$, then the message m is replaced by a uniform message m^* of the same length which is signcrypted into s using $\mathtt{P}_i$'s sender private key and $\mathtt{P}_j$'s receiver public key, and $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ is output at $\mathtt{E}$. Note that if $i = S$ (that is, the sender is the designated sender $\mathtt{P}_S$), then $\mathcal{A}_t$ uses the provided signcryption oracle. In any case, the mapping $((s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}) \mapsto m)$ is stored into a table M for later reference.

- If $\omega^{-1}(i,j) > t$, then the message m is signcrypted into s using $\mathtt{P}_i$'s sender private key and $\mathtt{P}_j$'s receiver public key, and $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ is output at $\mathtt{E}$. Note that if $i = S$ (that is, the sender is the designated sender $\mathtt{P}_S$), then $\mathcal{A}_t$ uses the provided signcryption oracle.
- If $\omega^{-1}(i,j) = t$ (that is, the parties are exactly the designated sender $\mathtt{P}_S$ and receiver $\mathtt{P}_R$), then the message m is signcrypted into s using the provided signcryption oracle, and $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ is output at $\mathtt{E}$. Moreover, in this case the mapping $((s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}) \mapsto m)$ is stored into table M.

On input $(\mathtt{inject}, s, \mathsf{ID}, \mathsf{ID}')$ **at interface** $\mathtt{E}$**:** The reduction $\mathcal{A}_t$ retrieves $\mathsf{ID}_{\mathtt{P}_i}$ from ID and $\mathtt{P}_j$ from ID' (recall that $i, j \in [n]$), and if both parties have previously successfully registered, $\mathcal{A}_t$ performs the following case distinction:

- If $(i,j) \notin \mathcal{L}$, then the signcryptext s is unsigncrypted into m using $\mathtt{P}_j$'s receiver private key and $\mathtt{P}_i$'s sender public key, and (m, ID) is output at $\mathtt{P}_j$.
- If $(i,j) \in \mathcal{L}$, then the further case distinction is made:
  - If $\omega^{-1}(i,j) < t$, then, if possible, the corresponding value m is retrieved from table M and (m, ID) is output at $\mathtt{P}_j$. In case no mapping exists, no output is produced for interface $\mathtt{P}_j$.
  - If $\omega^{-1}(i,j) > t$, then the signcryptext s is unsigncrypted into m using $\mathtt{P}_j$'s receiver private key and $\mathtt{P}_i$'s sender public key, and (m, ID) is output at $\mathtt{P}_j$. Note that if $j = R$ (that is, the receiver is the designated receiver $\mathtt{P}_R$), then $\mathcal{A}_t$ uses the provided unsigncryption oracle.
  - If $\omega^{-1}(i,j) = t$ (that is, the parties are exactly the designated sender $\mathtt{P}_S$ and receiver $\mathtt{P}_R$), then if possible the corresponding value m is retrieved from table M, otherwise the signcryptext s is unsigncrypted into m using the provided unsigncryption oracle. If a value $m \neq \perp$ can be obtained this way, (m, ID) is output at $\mathtt{P}_j$ and otherwise, no output is produced.

Towards a standard hybrid argument, note that:

- $\Pr\left[\mathcal{A}_1^{\mathbf{Real}_\Psi^{\mathsf{MOS}}} = 1\right] = \Pr\left[\mathbf{DH}_0^{\mathcal{Z}} = 1\right]$, that is, if the reduction adversary is connected to real oracles, and it sets (S, R) as the first pair of indexes in $\mathcal{L}$ according to the ordering induced by ω, then for the distinguisher $\mathbf{D}$ the view is the same as if it was connected to the real world resource $\mathbf{H}_0^{\mathcal{Z}}$, since all pairs of parties with indexes after (S, R) as well as $(\mathtt{P}_S, \mathtt{P}_R)$ act as real users.

- $\Pr\left[\mathcal{A}_{\ell^2}^{\mathbf{Ideal}_\Psi^{\mathsf{MOS}}} = 1\right] = \Pr\left[\mathbf{DH}_1^{\mathcal{Z}} = 1\right]$, that is, if the reduction adversary is connected to ideal oracles, and it sets (S, R) as the last pair of indexes in $\mathcal{L}$ according to the ordering induced by ω, then for the distinguisher $\mathbf{D}$ the view is the same as if it was connected to the ideal world resource $\mathbf{H}_1^{\mathcal{Z}}$, since all pairs of parties with indexes before (S, R) as well as $(\mathtt{P}_S, \mathtt{P}_R)$ act as ideal users. In particular, upon an input $(\mathtt{inject}, \cdot, \cdot, \cdot)$, if no corresponding input was given to the system before, no message is output, i.e., in both systems, even if the condition of bad_1 would be satisfied, both system define the resulting plaintext to be $\perp$.

- $\Pr\left[\mathcal{A}_t^{\mathbf{Ideal}_\Psi^{\mathsf{MOS}}} = 1\right] = \Pr\left[\mathcal{A}_{t+1}^{\mathbf{Real}_\Psi^{\mathsf{MOS}}} = 1\right]$, that is, if the reduction adversary $\mathcal{A}_t$ is connected to ideal oracles, then for the distinguisher $\mathbf{D}$ the view is the same as if it was being used by the reduction adversary $\mathcal{A}_{t+1}$ when connected to real oracles.

We can now conclude the proof using a standard hybrid argument:

$$\mathsf{Adv}^{\mathsf{MOS}}_{\Psi,\mathcal{A}} = \sum_{t=1}^{\ell^2} \mathsf{Adv}^{\mathsf{MOS}}_{\Psi,\mathcal{A}_t} \cdot \Pr[T=t] \tag{5.2}$$

$$= \frac{1}{\ell^2} \sum_{t=1}^{\ell^2} \left(\Pr\left[\mathcal{A}_t^{\mathbf{Real}^{\mathsf{MOS}}_{\Psi}} = 1\right] - \Pr\left[\mathcal{A}_t^{\mathbf{Ideal}^{\mathsf{MOS}}_{\Psi}} = 1\right] \right) \tag{5.3}$$

$$= \frac{1}{\ell^2} \sum_{t=1}^{\ell^2} \left(\Pr\left[\mathcal{A}_t^{\mathbf{Real}^{\mathsf{MOS}}_{\Psi}} = 1\right] - \Pr\left[\mathcal{A}_{t+1}^{\mathbf{Real}^{\mathsf{MOS}}_{\Psi}} = 1\right] \right) \tag{5.4}$$

$$= \frac{1}{\ell^2} \left(\Pr\left[\mathcal{A}_1^{\mathbf{Real}^{\mathsf{MOS}}_{\Psi}} = 1\right] - \Pr\left[\mathcal{A}_{\ell^2+1}^{\mathbf{Real}^{\mathsf{MOS}}_{\Psi}} = 1\right] \right) \tag{5.5}$$

$$= \frac{1}{\ell^2} \left(\Pr\left[\mathcal{A}_1^{\mathbf{Real}^{\mathsf{MOS}}_{\Psi}} = 1\right] - \Pr\left[\mathcal{A}_{\ell^2}^{\mathbf{Ideal}^{\mathsf{MOS}}_{\Psi}} = 1\right] \right) \tag{5.6}$$

$$= \frac{1}{\ell^2} \left(\Pr\left[\mathbf{DH}_0^{\mathcal{Z}} = 1\right] - \Pr\left[\mathbf{DH}_1^{\mathcal{Z}} = 1\right] \right) \tag{5.7}$$

$$= \frac{1}{\ell^2} \cdot \Delta^{\mathbf{D}}(\mathbf{H}_0^{\mathcal{Z}}, \mathbf{H}_1^{\mathcal{Z}}), \tag{5.8}$$

where for (5.2) we used $\mathcal{A} = \mathcal{A}_T$ and the law of total probability, for (5.3) we used $\Pr[T=t] = \frac{1}{\ell^2}$ (for any $t \in [\ell^2]$), for (5.4), (5.6), and (5.7) we used the three equalities outlined above, for (5.5) we used the standard hybrid argument, and for (5.8) we used the definition of the distinguishing advantage. This proves that systems $\mathbf{H}_0^{\mathcal{Z}}$ and $\mathbf{H}_1^{\mathcal{Z}}$ are computationally indistinguishable, that is, for any distinguisher $\mathbf{D}$,

$$\Delta^{\mathbf{D}}(\mathbf{H}_0^{\mathcal{Z}}, \mathbf{H}_1^{\mathcal{Z}}) \leq n^2 \cdot \mathsf{Adv}^{\mathsf{MOS}}_{\Psi,\rho_{\mathcal{Z}}(\mathbf{D})},$$

since $\ell \leq n$. ◊

Finally, we get the reduction stated in Theorem 5.3.1 by formally defining $\rho_1(\mathbf{D}) \coloneqq \arg\max_{\rho_{\mathcal{Z}}(\mathbf{D}):\mathcal{Z}\subseteq\mathcal{U}\setminus\{\mathtt{E}\}} \{\mathsf{Adv}^{\mathsf{MOS}}_{\Psi,\rho_{\mathcal{Z}}(\mathbf{D})}\}$, where we assume that the return value is a single adversary (note that there will always be a maximum in this finite set of possibilities and in case of multiple maxima, we simple apply a tie-breaking rule like lexicographic ordering).

3.) The third hybrid system $\mathbf{H}_2^{\mathcal{Z}}$ is depicted in Figure 5.12 and is a slight modification of the second: in this hybrid system, a message

cannot be injected for pairs of parties, where the source of a message is a honest party whose key was not stolen (and the recipient is a party about whom we do not make an assumption). The variable bad_2 is true, if and only if the adversary succeeds in breaking the security between any such pair.

For the difference between the second and third hybrid, we can prove the following claim.

Claim 2. *For any distinguisher* $\mathbf{D}$ *we have* $\Delta^{\mathbf{D}}(\mathbf{H}_1^{\mathcal{Z}}, \mathbf{H}_2^{\mathcal{Z}}) \leq n \cdot \mathsf{Adv}^{\mathsf{MIS\text{-}Auth}}_{\Psi,\rho_{\mathcal{Z}}(\mathbf{D})}$, *that is, a (successful) distinguisher* $\mathbf{D}$ *for* $\mathbf{H}_1^{\mathcal{Z}}$ *and* $\mathbf{H}_2^{\mathcal{Z}}$ *can be transformed into a (successful) forger* $\mathcal{A} = \rho_{\mathcal{Z}}(\mathbf{D})$ *(defined in the proof) for* $\mathbf{Auth}^{\mathsf{MIS}}_{\Psi}$.

Proof of claim. The idea is again to use a standard hybrid argument, but this time on the set of all senders which did not leak their secret key in the real world, that is, the set of all users which in the ideal world can send messages in an authentic fashion to other users. More precisely, we select one of those users uniformly at random, and for his (outgoing) communications we use the oracles provided to the adversary by the security experiment. Note that for the very special case that $\mathcal{Z} = \emptyset$ the statement is trivial since there is no difference between $\mathbf{H}_1^{\mathcal{Z}}$ and $\mathbf{H}_2^{\mathcal{Z}}$ by definition, hence, we assume $\mathcal{Z} \neq \emptyset$ in the sequel.

Let the system $\mathbf{D}$ be a distinguisher for $\mathbf{H}_1^{\mathcal{Z}}$ and $\mathbf{H}_2^{\mathcal{Z}}$, and let $\mathcal{H} = \{i \in [n] \,|\, \mathtt{M}_i \notin \mathcal{Z}\}$, $\ell = |\mathcal{H}|$, be as defined in the proof of Claim 1. We again assume an order over $\mathcal{H}$, that is, we again fix some efficiently computable bijection $\omega : [\ell] \to \mathcal{H}$ as well as its efficiently computable inverse map $\omega^{-1} : \mathcal{H} \to [\ell]$. We construct an adversary $\mathcal{A}$ for winning $\mathbf{Auth}^{\mathsf{MIS}}_{\Psi}$ using distinguisher $\mathbf{D}$ via a reduction $\rho_{\mathcal{Z}}(\cdot)$, denoted $\mathcal{A} = \rho_{\mathcal{Z}}(\mathbf{D})$.

The reduction works similarly to the one in Claim 1, i.e., by first choosing an index t uniformly at random from $[\ell]$, and then computing the index $S = \omega(t)$, $S \in [n]$, of a *designated sender* $\mathtt{P}_S$. Again, let $\mathcal{A}_t$ be the same as $\mathcal{A}$ but where the index t is fixed instead of uniformly randomly selected. For a random variable T uniformly distributed over $[\ell]$, this implies $\mathcal{A} = \mathcal{A}_T$. Recall that when $\mathcal{A}_t$ interacts with $\mathbf{Auth}^{\mathsf{MIS}}_{\Psi}$, it receives a sender public key $pk^{\star}_S$, which will be set as $\mathtt{P}_S$'s receiver public key. As before, upon honest registration of a new entity, $\mathcal{A}_t$ stores a sender key-pair (generated using Gen_S) and a receiver key-pair (generated using the oracle $\mathtt{Gen}$ provided by $\mathbf{Auth}^{\mathsf{MIS}}_{\Psi}$), except that for party $\mathtt{P}_S$ only a receiver key-pair is generated (recall that for $\mathtt{P}_S$, $\mathcal{A}_t$ uses

the sender public key provided by $\mathbf{Auth}_{\Psi}^{\mathsf{MIS}}$, whereas the corresponding sender secret key is "hard-coded" into the provided signcryption oracle). In any case, the mapping between identity and public keys is recorded. Upon registration directly at interface $\mathtt{E}$, the reduction remembers the corresponding mapping between identity and public keys. Whenever $\mathtt{reveal}$ is input at interface $\mathtt{M}_i \in \mathcal{Z}$ (i.e., $i \notin \mathcal{H}$), $\mathcal{A}_t$ returns the two generated key-pairs to the distinguisher $\mathbf{D}$ (and $\perp$ if $\mathtt{M}_i \notin \mathcal{Z}$, i.e., $i \in \mathcal{H}$) and on $\mathtt{fetchAll}$ the recorded mapping between identities and public keys is returned. We now describe the behavior of $\mathcal{A}_t$ on the remaining inputs.

On input $(\mathtt{send}, m, \mathsf{ID})$ **at interface** $\mathtt{P}_i$**:** The reduction $\mathcal{A}_t$ retrieves $\mathsf{ID}_{\mathtt{P}_i}$ and $\mathtt{P}_j$ from ID (recall that $i, j \in [n]$), and if both parties have previously successfully registered, $\mathcal{A}_t$ performs the following case distinction:

- If $i \notin \mathcal{H}$, then the message m is signcrypted into s using $\mathtt{P}_i$'s sender private key and $\mathtt{P}_j$'s receiver public key, and $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ is output at $\mathtt{E}$.
- If $i \in \mathcal{H}$, then the further case distinction is made. First, if $(i, j) \in \mathcal{L}$ then the message m is first replaced replaced by a uniform message m^* of the same length. Otherwise, m is left untouched. Continue with the remaining two conditions:
  - If $\omega^{-1}(i) \neq t$, then the message m is signcrypted into s using $\mathtt{P}_i$'s sender private key and $\mathtt{P}_j$'s receiver public key, and $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ is output at $\mathtt{E}$. The mapping $((s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}) \mapsto m)$ is stored into a table M for later reference.
  - If $\omega^{-1}(i) = t$ (that is, the party $\mathtt{P}_i$ is exactly the designated sender $\mathtt{P}_S$), then the message m is signcrypted into s using the provided signcryption oracle, and $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ is then output at $\mathtt{E}$. The mapping $((s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}) \mapsto m)$ is stored into a table M for later reference.

On input $(\mathtt{inject}, s, \mathsf{ID}, \mathsf{ID}')$ **at interface** $\mathtt{E}$**:** The reduction $\mathcal{A}_t$ retrieves $\mathsf{ID}_{\mathtt{P}_i}$ from ID and $\mathtt{P}_j$ from ID' (recall that $i, j \in [n]$), and if both parties have previously successfully registered, $\mathcal{A}_t$ performs the following case distinction:

- If $i \notin \mathcal{H}$, then the signcryptext s is unsigncrypted into m using $\mathtt{P}_j$'s receiver private key and $\mathtt{P}_i$'s sender public key, and (m, ID) is output at $\mathtt{P}_j$.
- If $i \in \mathcal{H}$, then the further case distinction is made:
  - If $(i, j) \in \mathcal{L}$ **or** $\omega^{-1}(i) < t$ then, if possible, the corresponding value m is retrieved from table M and (m, ID) is output at $\mathtt{P}_j$. In case no mapping exists, no output is produced for interface $\mathtt{P}_j$.
  - If $\omega^{-1}(i) > t$ and $j \notin \mathcal{H}$, then the signcryptext s is unsigncrypted into m using $\mathtt{P}_j$'s receiver private key and $\mathtt{P}_i$'s sender public key, and (m, ID) is output at $\mathtt{P}_j$.
  - If $\omega^{-1}(i) = t$ (that is, the party $\mathtt{P}_i$ is exactly the designated sender $\mathtt{P}_S$) and $j \notin \mathcal{H}$, then the signcryptext s is unsigncrypted into m using the provided unsigncryption oracle, and (m, ID) is output at $\mathtt{P}_j$.

Note that we can naturally define more fine-grained hybrid systems $\mathbf{H}^{\mathcal{Z}}_{1,t}$ and $\mathbf{H}^{\mathcal{Z}}_{2,t}$, for $t \in [\ell]$, such that $\mathbf{H}^{\mathcal{Z}}_{2,t}$ and $\mathbf{H}^{\mathcal{Z}}_{1,t+1}$ are exactly the system that $\mathcal{A}_t$ emulates to $\mathbf{D}$. In particular the flag win in $\mathbf{Auth}^{\mathsf{MIS}}_{\Psi}$ is set if and only if the flag bad_2 is set upon successfully injecting a forged signcryptext on behalf of $\omega(t)$. For sake of clarity, we will refer to this new flag by bad^t_2. Moreover, systems $H^{\mathcal{Z}}_{1,t}$ and $\mathbf{H}^{\mathcal{Z}}_{2,t}$ have an equivalent behavior unless this flag is set. Since the behavior of the fine-grained hybrids is analogous to the one described above for $\mathcal{A}_t$, we refrain from formally describing $\mathbf{H}^{\mathcal{Z}}_{1,t}$ and $\mathbf{H}^{\mathcal{Z}}_{2,t}$. Towards a standard hybrid argument, note that:

- $\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{1,1} = 1\right] = \Pr\left[\mathbf{DH}^{\mathcal{Z}}_{1} = 1\right]$, that is, the view of distinguisher $\mathbf{D}$ when interacting with $\mathbf{H}^{\mathcal{Z}}_{1,1}$ or $\mathbf{H}^{\mathcal{Z}}_{1}$ is the same.
- $\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{2,\ell} = 1\right] = \Pr\left[\mathbf{DH}^{\mathcal{Z}}_{2} = 1\right]$, that is, the view of distinguisher $\mathbf{D}$ when interacting with $\mathbf{H}^{\mathcal{Z}}_{2,\ell}$ or $\mathbf{H}^{\mathcal{Z}}_{2}$ is the same.
- $\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{2,t} = 1\right] = \Pr\left[\mathbf{DH}^{\mathcal{Z}}_{1,t+1} = 1\right]$, that is, the view of distinguisher $\mathbf{D}$ when interacting with $\mathbf{H}^{\mathcal{Z}}_{2,t}$ or $\mathbf{H}^{\mathcal{Z}}_{1,t+1}$ is the same.

We can now conclude the proof:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{MIS\text{-}Auth}}_{\Psi,\mathcal{A}} &= \frac{1}{\ell}\sum_{t=1}^{\ell}\Pr\left[\mathcal{A}_t^{\mathbf{Auth}^{\mathsf{MIS}}_{\Psi}} \text{ sets } \mathsf{win}\right] && (5.9)\\
&= \frac{1}{\ell}\sum_{t=1}^{\ell}\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{2,t} \text{ sets } \mathsf{bad}^t_2\right] && (5.10)\\
&\geq \frac{1}{\ell}\sum_{t=1}^{\ell}\left(\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{1,t}=1\right]-\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{2,t}=1\right]\right) && (5.11)\\
&= \frac{1}{\ell}\sum_{t=1}^{\ell}\left(\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{1,t}=1\right]-\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{1,t+1}=1\right]\right)\\
&= \frac{1}{\ell}\left(\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{1,1}=1\right]-\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{1,\ell+1}=1\right]\right)\\
&= \frac{1}{\ell}\left(\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{1,1}=1\right]-\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{2,\ell}=1\right]\right)\\
&= \frac{1}{\ell}\left(\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{1}=1\right]-\Pr\left[\mathbf{DH}^{\mathcal{Z}}_{2}=1\right]\right)\\
&= \frac{1}{\ell}\cdot\Delta^{\mathbf{D}}(\mathbf{H}^{\mathcal{Z}}_1,\mathbf{H}^{\mathcal{Z}}_2).
\end{aligned}$$

For (5.9) we used $\Pr[T=t]=\frac{1}{\ell}$ (for any $t\in[\ell]$) and the definition of the advantage of $\mathcal{A}_t$ when interacting with $\mathbf{Auth}^{\mathsf{MIS}}_{\Psi}$, for (5.10) we used the fact that $\mathcal{A}_t$ perfectly simulates $\mathbf{H}^{\mathcal{Z}}_{2,t}$ to $\mathbf{D}$. For (5.11) we used [Mau02, Theorem 1] (or equivalently, a concretization thereof for code-based games in [BR06, Lemma 2 (*"Fundameltal Lemma of Game-Playing"*)]) that relates game-winning and distinguishing. The remaining steps hold by the standard hybrid argument and the by the equalities outlined above. This proves that systems $\mathbf{H}^{\mathcal{Z}}_1$ and $\mathbf{H}^{\mathcal{Z}}_2$ are computationally indistinguishable, that is, for any distinguisher $\mathbf{D}$,

$$\Delta^{\mathbf{D}}(\mathbf{H}^{\mathcal{Z}}_1,\mathbf{H}^{\mathcal{Z}}_2)\leq n\cdot\mathsf{Adv}^{\mathsf{MIS\text{-}Auth}}_{\Psi,\rho_{\mathcal{Z}}(\mathbf{D})},$$

since $\ell\leq n$. ◊

Finally, the reduction $\rho_2(\cdot)$ stated in the theorem is defined analogously to the reduction $\rho_1(\cdot)$ above.

4.) The fourth hybrid system $\mathbf{H}_3^{\mathcal{Z}}$, first depicted in Figure 5.13, is a slight modification of the third, where we replace encryptions of messages by encryptions of random messages in case the recipient of a message is an honest party whose key is not compromised (and the source is a party about whom we make no assumption).

For the difference between the third and the fourth hybrid, we can prove the following claim.

Claim 3. *For any distinguisher* $\mathbf{D}$ *we have* $\Delta^{\mathbf{D}}(\mathbf{H}_2^{\mathcal{Z}}, \mathbf{H}_3^{\mathcal{Z}}) \leq n \cdot \mathsf{Adv}^{\mathsf{MIS\text{-}Conf}}_{\Psi,\rho_{\mathcal{Z}}(\mathbf{D})}$*, that is, a (successful) distinguisher* $\mathbf{D}$ *for* $\mathbf{H}_2^{\mathcal{Z}}$ *and* $\mathbf{H}_3^{\mathcal{Z}}$ *can be transformed into a (successful) distinguisher* $\mathcal{A} = \rho_{\mathcal{Z}}(\mathbf{D})$ *(defined in the proof) for* $\mathbf{Real}_{\Psi}^{\mathsf{MIS\text{-}Conf}}$ *and* $\mathbf{Ideal}_{\Psi}^{\mathsf{MIS\text{-}Conf}}$*.*

Proof of claim. The idea is again to use a standard hybrid argument, but this time on the set of all receivers which did not leak their secret key in the real world, that is, the set of all users which in the ideal world can receive messages in a confidential fashion. More precisely, we select one of those users uniformly at random, and for his (ingoing) communications we use the oracles provided to the adversary by the security experiment (again, the reduction will depend on the corruption set $\mathcal{Z}$). Note that for the very special case that $\mathcal{Z} = \emptyset$ the statement is trivial since there is no difference between $\mathbf{H}_2^{\mathcal{Z}}$ and $\mathbf{H}_3^{\mathcal{Z}}$ by definition. So, we assume $\mathcal{Z} \neq \emptyset$ in the sequel.

Similarly to above, let the system $\mathbf{D}$ be a distinguisher for $\mathbf{H}_2^{\mathcal{Z}}$ and $\mathbf{H}_3^{\mathcal{Z}}$, and let $\mathcal{H} = \{i \in [n] \,|\, \mathtt{M}_i \notin \mathcal{Z}\}$ be the set of indexes of uncorrupted parties, with $\ell = |\mathcal{H}|$. Let us also fix an order over $\mathcal{H}$, that is, fix some efficiently computable bijection $\omega : [\ell] \to \mathcal{H}$ as well as its efficiently computable inverse map $\omega^{-1} : \mathcal{H} \to [\ell]$. We construct an adversary $\mathcal{A}$ for distinguishing $\mathbf{Real}_{\Psi}^{\mathsf{MIS\text{-}Conf}}$ from $\mathbf{Ideal}_{\Psi}^{\mathsf{MIS\text{-}Conf}}$ using distinguisher $\mathbf{D}$ via a reduction $\rho_{\mathcal{Z}}(\cdot)$, denoted $\mathcal{A} = \rho_{\mathcal{Z}}(\mathbf{D})$.

The reduction works by first choosing an index t uniformly at random from $[\ell]$, and then computing the index $R = \omega(t)$, $R \in [n]$, of a *designated receiver* $\mathtt{P}_R$. In the following, let $\mathcal{A}_t$ be the same as $\mathcal{A}$ but where the index t is fixed instead of uniformly randomly selected. For a random variable T uniformly distributed over $[\ell]$, this implies $\mathcal{A} = \mathcal{A}_T$. Recall that when $\mathcal{A}_t$ interacts with $\mathbf{Real}_{\Psi}^{\mathsf{MIS\text{-}Conf}}$ or $\mathbf{Ideal}_{\Psi}^{\mathsf{MIS\text{-}Conf}}$, it receives a receiver public key $pk_R^{\star}$, which will be set as $\mathtt{P}_R$'s receiver public key. Upon honest registration, $\mathcal{A}_t$ stores for each user both a receiver key-pair

(generated using Gen_R) and a sender key-pair (generated using the oracle $\mathtt{Gen}$ provided by either $\mathbf{Real}_\Psi^{\mathsf{MIS\text{-}Conf}}$ or $\mathbf{Ideal}_\Psi^{\mathsf{MIS\text{-}Conf}}$), except that for party $\mathtt{P}_R$ only a sender key-pair is generated (recall that for $\mathtt{P}_R$, $\mathcal{A}_t$ uses the receiver public key provided by either $\mathbf{Real}_\Psi^{\mathsf{MIS\text{-}Conf}}$ or $\mathbf{Ideal}_\Psi^{\mathsf{MIS\text{-}Conf}}$, whereas the corresponding receiver secret key is "hard-coded" into the provided unsigncryption oracle). The reduction records the mapping of identities to public keys. Dishonest registrations are handled as in the proof of Claim 1, as well as answers to `fetchAll` queries. Whenever `reveal` is input at interface $\mathtt{M}_i \in \mathcal{Z}$ (i.e., $i \notin \mathcal{H}$), $\mathcal{A}_t$ returns the two generated key-pairs to the distinguisher $\mathbf{D}$ (and $\perp$ if $\mathtt{M}_i \notin \mathcal{Z}$, i.e., $i \in \mathcal{H}$). We now describe the behavior of $\mathcal{A}_t$ on the remaining inputs.

On input $(\mathtt{send}, m, \mathsf{ID})$ **at interface** $\mathtt{P}_i$**:** The reduction $\mathcal{A}_t$ retrieves $\mathsf{ID}_{\mathtt{P}_i}$ and $\mathtt{P}_j$ from ID (recall that $i, j \in [n]$), and if both parties have previously successfully registered, $\mathcal{A}_t$ performs the following case distinction:

- If $j \notin \mathcal{H}$, then the message m is signcrypted into s using $\mathtt{P}_i$'s sender private key and $\mathtt{P}_j$'s receiver public key, and $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ is output at $\mathtt{E}$.
- If $j \in \mathcal{H}$, then the further case distinction is made:
  - If $(i, j) \in \mathcal{L}$ **or** $\omega^{-1}(j) < t$, then the message m is replaced by a uniform message m^* of the same length which is signcrypted into s using $\mathtt{P}_i$'s sender private key and $\mathtt{P}_j$'s receiver public key, and $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ is output at $\mathtt{E}$.
  - Else if $\omega^{-1}(j) > t$, then the message m is signcrypted into s using $\mathtt{P}_i$'s sender private key and $\mathtt{P}_j$'s receiver public key, and $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ is output at $\mathtt{E}$.
  - Else if $\omega^{-1}(j) = t$ (that is, the party $\mathtt{P}_j$ is exactly the designated receiver $\mathtt{P}_R$) (and $i \notin \mathcal{H}$), then the message m is signcrypted into s using the provided signcryption oracle, by providing as input the key-pair of the sender $\mathtt{P}_i$ and the receiver public key $pk_R^\star$. $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ is then output at $\mathtt{E}$.

  In any of the above cases, the mapping $((s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}) \mapsto m)$ is stored into a table M for later reference.

On input $(\texttt{inject}, s, \mathsf{ID}, \mathsf{ID}')$ **at interface** $\texttt{E}$**:** The reduction $\mathcal{A}_t$ retrieves $\mathsf{ID}_{\texttt{P}_i}$ from ID and $\texttt{P}_j$ from ID' (recall that $i, j \in [n]$), and if both parties have previously successfully registered, $\mathcal{A}_t$ performs the following case distinction:

- If $i \in \mathcal{H}$ then, if possible, the corresponding value m is retrieved from table M and (m, ID) is output at $\texttt{P}_j$. In case no mapping exists, no output is produced for interface $\texttt{P}_j$.
- If $i \notin \mathcal{H}$, then the further case distinction is made:
  - If $\omega^{-1}(j) < t$ then if possible the corresponding value m is retrieved from table M, otherwise the signcryptext s is unsigncrypted into m using $\texttt{P}_j$'s receiver private key and $\texttt{P}_i$'s sender public key. If a value $m \neq \perp$ can be obtained this way, (m, ID) is output at $\texttt{P}_j$ and otherwise, no output is produced.
  - If $\omega^{-1}(j) > t$ and $j \notin \mathcal{H}$, then the signcryptext s is unsigncrypted into m using $\texttt{P}_j$'s receiver private key and $\texttt{P}_i$'s sender public key, and (m, ID) is output at $\texttt{P}_j$.
  - If $\omega^{-1}(j) = t$ (that is, the party $\texttt{P}_i$ is exactly the designated sender $\texttt{P}_S$) and $j \notin \mathcal{H}$, then if possible the corresponding value m is retrieved from table M, otherwise the signcryptext s is unsigncrypted into m using the provided unsigncryption oracle. If a value $m \neq \perp$ can be obtained this way, (m, ID) is output at $\texttt{P}_j$ and otherwise, no output is produced.

Towards a hybrid argument, note that:

- $\Pr\left[\mathcal{A}_1^{\mathbf{Real}_\Psi^{\mathsf{MIS\text{-}Conf}}} = 1\right] = \Pr\left[\mathbf{DH}_2^{\mathcal{Z}} = 1\right]$, that is, if the reduction adversary is connected to real oracles, and it sets R as the first index in $\mathcal{H}$ according to the ordering induced by ω, then for the distinguisher $\mathbf{D}$ the view is the same as if it was connected to the real world resource $\mathbf{H}_2^{\mathcal{Z}}$, since for all parties with index $j \in \mathcal{H}$ and greater t as well as $\texttt{P}_t$ confidentiality is only enforced for messages originating form an honest sender. Note that injecting messages in the name of corrupted senders is possible (as in a typical CCA-style game).

- $\Pr\left[\mathcal{A}_\ell^{\mathbf{Ideal}_\Psi^{\mathsf{MIS\text{-}Conf}}} = 1\right] = \Pr\left[\mathbf{DH}_3^{\mathcal{Z}} = 1\right]$, that is, if the reduction adversary is connected to ideal oracles, and it sets R as the last index in $\mathcal{H}$ according to the ordering induced by ω, then for the distinguisher $\mathbf{D}$ the view is the same as if it was connected to the ideal world resource $\mathbf{H}_3^{\mathcal{Z}}$, since now, for all uncorrupted parties with index $j \in \mathcal{H}$ and before R as well as P_R confidentiality of the message is enforced, even for messages originating from dishonest senders. However, injecting message in the name of corrupted users is still possible.
- $\Pr\left[\mathcal{A}_t^{\mathbf{Ideal}_\Psi^{\mathsf{MIS\text{-}Conf}}} = 1\right] = \Pr\left[\mathcal{A}_{t+1}^{\mathbf{Real}_\Psi^{\mathsf{MIS\text{-}Conf}}} = 1\right]$, that is, if the reduction adversary $\mathcal{A}_t$ is connected to ideal oracles, then for the distinguisher $\mathbf{D}$ the view is the same as if it was being used by the reduction adversary $\mathcal{A}_{t+1}$ when connected to real oracles, since in the former case, the ideal oracles enforce confidentiality (by encryption a random message) and in the latter, this happens by definition of the reduction.

We can now conclude the proof using the hybrid argument:

$$
\begin{aligned}
\mathsf{Adv}_{\Psi,\mathcal{A}}^{\mathsf{MIS\text{-}Conf}} &= \sum_{t=1}^{\ell} \mathsf{Adv}_{\Psi,\mathcal{A}_t}^{\mathsf{MIS\text{-}Conf}} \cdot \Pr[T = t] \\
&= \frac{1}{\ell} \sum_{t=1}^{\ell} \left(\Pr\left[\mathcal{A}_t^{\mathbf{Real}_\Psi^{\mathsf{MIS\text{-}Conf}}} = 1\right] - \Pr\left[\mathcal{A}_t^{\mathbf{Ideal}_\Psi^{\mathsf{MIS\text{-}Conf}}} = 1\right] \right) \\
&= \frac{1}{\ell} \sum_{t=1}^{\ell} \left(\Pr\left[\mathcal{A}_t^{\mathbf{Real}_\Psi^{\mathsf{MIS\text{-}Conf}}} = 1\right] - \Pr\left[\mathcal{A}_{t+1}^{\mathbf{Real}_\Psi^{\mathsf{MIS\text{-}Conf}}} = 1\right] \right) \\
&= \frac{1}{\ell} \left(\Pr\left[\mathcal{A}_1^{\mathbf{Real}_\Psi^{\mathsf{MIS\text{-}Conf}}} = 1\right] - \Pr\left[\mathcal{A}_{\ell+1}^{\mathbf{Real}_\Psi^{\mathsf{MIS\text{-}Conf}}} = 1\right] \right) \\
&= \frac{1}{\ell} \left(\Pr\left[\mathcal{A}_1^{\mathbf{Real}_\Psi^{\mathsf{MIS\text{-}Conf}}} = 1\right] - \Pr\left[\mathcal{A}_\ell^{\mathbf{Ideal}_\Psi^{\mathsf{MIS\text{-}Conf}}} = 1\right] \right) \\
&= \frac{1}{\ell} \left(\Pr\left[\mathbf{DH}_2^{\mathcal{Z}} = 1\right] - \Pr\left[\mathbf{DH}_3^{\mathcal{Z}} = 1\right] \right) \\
&= \frac{1}{\ell} \cdot \Delta^{\mathbf{D}}(\mathbf{H}_2^{\mathcal{Z}}, \mathbf{H}_3^{\mathcal{Z}}),
\end{aligned}
$$

where the steps follow analogously to the proof of Claim 1. Hence, systems $\mathbf{H}_2^{\mathcal{Z}}$ and $\mathbf{H}_3^{\mathcal{Z}}$ are computationally indistinguishable, that is, for

any distinguisher $\mathbf{D}$,

$$\Delta^{\mathbf{D}}(\mathbf{H}_2^{\mathcal{Z}}, \mathbf{H}_3^{\mathcal{Z}}) \leq n \cdot \mathsf{Adv}^{\mathsf{MIS\text{-}Conf}}_{\Psi,\rho_{\mathcal{C}}(\mathbf{D})},$$

since $\ell \leq n$. ◊

Finally, the reduction $\rho_3(\cdot)$ stated in the theorem is defined analogously to the reduction $\rho_1(\cdot)$ above.

The final four steps are completed in the supplementary material. We give a brief overview:

5.) The hybrid system $\mathbf{H}_4^{\mathcal{Z}}$ is a syntactic modification of the previous one and detailed in Appendix A.1.1.

6.) The hybrid system $\mathbf{H}_5^{\mathcal{Z}}$ is a syntactic modification of the previous one and detailed in Appendix A.1.2.

7.) The hybrid system $\mathbf{H}_6^{\mathcal{Z}}$ is a syntactic modification of the previous one and detailed in Appendix A.1.3.

8.) We show how hybrid system $\mathbf{H}_6^{\mathcal{Z}}$ can be seen as the composition of the secure network and the simulators and the details are given in Appendix A.1.4.

Since $\mathbf{H}_4^{\mathcal{Z}} = \mathbf{H}_5^{\mathcal{Z}} = \mathbf{H}_6^{\mathcal{Z}}$, the latter being equivalent to the ideal world, this concludes the argument and the proof of Theorem 5.3.1. □

A special case. An interesting corollary that we can directly observe by looking at the game-hopping argument is that in the special case when the set of interfaces with potential dishonest behavior is the set $\{\mathtt{E}\}$, we get the following statement: The outsider security model implies the construction of a secure network if no honest parties' keys are compromised.

Corollary 5.3.2. *If there are no key compromises, i.e, $\mathcal{U} = \{\mathtt{E}\}$, then*

$$[\mathbf{Net}_n, \mathbf{CA}_n, \mathbf{Mem}_n]_{\phi^{\mathrm{real}}} \overset{(\pi^{\Psi}, \varepsilon, \{\mathtt{E}\})}{\Longrightarrow} \mathbf{SecNT}_{n\,\phi^{\mathrm{ideal}}},$$

for $\varepsilon(\mathbf{D}) := n^2 \cdot \mathsf{Adv}^{\mathsf{MOS}}_{\Psi,\rho_1(\mathbf{D})}$, where the reduction ρ_1 is defined in the proof of Claim 1.

Proof. The proof follows by the above arguments for $\mathcal{Z} := \emptyset$. □

Specification of hybrids as pseudo-code. On the following pages, we provide the formal specifications underlying our game-hopping argument for Claims 1 to 3. The remaining hybrids are found in the supplementary material Appendix A.

In the title of each box that depicts two hybrids at once, there are typically two names surrounded by solid or dashed boxes such as $\boxed{\mathbf{H}_0^{\mathcal{C}}}$ and $\boxed{\mathbf{H}_1^{\mathcal{C}}}$. This means that all code specifically surrounded by a dashed line is executed in $\mathbf{H}_0^{\mathcal{C}}$, but not $\mathbf{H}_1^{\mathcal{C}}$. Similarly, all code specifically surrounded by a solid line is executed in $\mathbf{H}_1^{\mathcal{C}}$, but not in $\mathbf{H}_0^{\mathcal{C}}$. All remaining code is executed in both systems. In cases where the box represents just one hybrid system, we might draw boxes to highlight certain parts of the code. The descriptions of the hybrid systems are given on the following pages.

The specification of the hybrid systems concludes our study on signcryption and closes the part on secure communication primitives.

Resource $\mathbf{H}_0^{\mathcal{Z}}$

Initialization

$J_{ca}, J_{net}, S \leftarrow \emptyset$
$I_1^{ca}, I_2^{ca}, I_1^{net}, I_2^{net}, T, L \leftarrow$ empty tables

Interface $\mathtt{P}_i$

Input: $(\mathtt{register}, \mathsf{ID})$
  if $\mathsf{ID}_{\mathtt{P}_i} = \perp$ **then**
    $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S$
    $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R$
    if $\mathsf{ID} \notin J_{ca}$ **then**
      $J_{ca} \leftarrow J_{ca} \cup \{\mathsf{ID}\}$
      $I_1^{ca}[\mathtt{P}_i] \leftarrow \mathsf{ID}$
      $I_2^{ca}[\mathsf{ID}] \leftarrow \mathtt{P}_i$
      $T[\mathsf{ID}] \leftarrow (pk_S, pk_R)$
      if $\mathsf{ID} \notin J_{net} \ \wedge \ I_1[\mathtt{P}_i] = \perp$ **then**
        $J_{net} \leftarrow J_{net} \cup \{\mathsf{ID}\}$
        $I_1^{net}[\mathtt{P}_i] \leftarrow \mathsf{ID}$
        $I_2^{net}[\mathsf{ID}] \leftarrow \mathtt{P}_i$
        $\mathsf{ID}_{\mathtt{P}_i} \leftarrow \mathsf{ID}$
        $val_{\mathtt{P}_i} \leftarrow [sk_S, pk_S, sk_R, pk_R]$
        if $\mathtt{M}_i \notin \mathcal{Z}$ **then**
          $S \leftarrow S \cup \{\mathsf{ID}\}$
        output Success at $\mathtt{P}_i$
      else
        output Fail at $\mathtt{P}_i$
    else
      output Fail at $\mathtt{P}_i$

Input: $(\mathtt{send}, m, \mathsf{ID})$
  if $\mathsf{ID}_{\mathtt{P}_i} \neq \perp$ **then**
    Let $val_{\mathtt{P}_i} = [sk_S, pk_S, sk_R, pk_R]$
    if $T[\mathsf{ID}] \neq \perp$ **then**
      Parse $T[\mathsf{ID}]$ as (pk'_S, pk'_R)
      if $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**
        $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m)$
        $L[s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}] \leftarrow m$
        output $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ at $\mathtt{E}$

Interface $\mathtt{E}$ **of Net**

Input: $(\mathtt{inject}, s, \mathsf{ID}_s, \mathsf{ID}_r)$
  if $\mathsf{ID}_r \in J_{net}$ **then**
    $\mathtt{P}_i \leftarrow I_2^{net}[\mathsf{ID}_r]$
    Let $val_{\mathtt{P}_i} = [sk_S, pk_S, sk_R, pk_R]$
    if $T[\mathsf{ID}_s] \neq \perp$ **then**
      Parse $T[\mathsf{ID}_s]$ as (pk'_S, pk'_R)
      if $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**
        $m \leftarrow \mathsf{Unsigncrypt}(sk_R, pk'_S, s)$
        if $(\mathsf{ID}_s, \mathsf{ID}_r) \in S \times S$ **then**
          $\mathsf{bad}_1 \leftarrow (m \neq \perp \wedge L[s, \mathsf{ID}_s, \mathsf{ID}_r] = \perp)$
        if $(\mathsf{ID}_s, \mathsf{ID}_r) \in S \times (J_{net} \setminus S)$ **then**
          $\mathsf{bad}_2 \leftarrow (m \neq \perp \wedge L[s, \mathsf{ID}_s, \mathsf{ID}_r] = \perp)$
        if $m \neq \perp$ **then**
          output (m, ID_s) at $\mathtt{P}_i$

Interface $\mathtt{E}$ **of CA**

Input: $(\mathtt{register}, \mathsf{ID}, val)$
  if $\mathsf{ID} \notin J$ **then**
    $T[\mathsf{ID}] \leftarrow val$
    $J_{ca} \leftarrow J_{ca} \cup \{\mathsf{ID}\}$
    output Success at $\mathtt{E}$
  else
    output Fail at $\mathtt{E}$

Input: `fetchAll`
  output (J, T) at $\mathtt{E}$

Interface $\mathtt{M}_i \in \mathcal{Z}$

Input: `reveal`
  output $val_{\mathtt{P}_i}$ at $\mathtt{M}_i$

Interface $\mathtt{M}_i \notin \mathcal{Z}$

Input: `reveal`
  output $\perp$ at $\mathtt{M}_i$

Figure 5.10: A concise description of the real world.

Resource $\boxed{\mathbf{H}_0^{\mathcal{Z}}}$ and $\boxed{\mathbf{H}_1^{\mathcal{Z}}}$

Initialization

$J_{ca}, J_{net}, S \leftarrow \emptyset$
$I_1^{ca}, I_2^{ca}, I_1^{net}, I_2^{net}, T, L \leftarrow$ empty tables

Interface $\mathtt{P}_i$

Input: $(\mathtt{register}, \mathsf{ID})$
  if $\mathsf{ID}_{\mathtt{P}_i} = \perp$ **then**
    $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S$
    $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R$
    if $\mathsf{ID} \notin J_{ca}$ **then**
      $J_{ca} \leftarrow J_{ca} \cup \{\mathsf{ID}\}$
      $I_1^{ca}[\mathtt{P}_i] \leftarrow \mathsf{ID}$
      $I_2^{ca}[\mathsf{ID}] \leftarrow \mathtt{P}_i$
      $T[\mathsf{ID}] \leftarrow (pk_S, pk_R)$
      if $\mathsf{ID} \notin J_{net} \ \wedge \ I_1[\mathtt{P}_i] = \perp$ **then**
        $J_{net} \leftarrow J_{net} \cup \{\mathsf{ID}\}$
        $I_1^{net}[\mathtt{P}_i] \leftarrow \mathsf{ID}$
        $I_2^{net}[\mathsf{ID}] \leftarrow \mathtt{P}_i$
        $\mathsf{ID}_{\mathtt{P}_i} \leftarrow \mathsf{ID}$
        $val_{\mathtt{P}_i} \leftarrow [sk_S, pk_S, sk_R, pk_R]$
        if $\mathtt{M}_i \notin \mathcal{Z}$ **then**
          $S \leftarrow S \cup \{\mathsf{ID}\}$
        output Success at $\mathtt{P}_i$
      else
        output Fail at $\mathtt{P}_i$
    else
      output Fail at $\mathtt{P}_i$

Input: $(\mathtt{send}, m, \mathsf{ID})$
  if $\mathsf{ID}_{\mathtt{P}_i} \neq \perp$ **then**
    Let $val_{\mathtt{P}_i} = [sk_S, pk_S, sk_R, pk_R]$
    if $T[\mathsf{ID}] \neq \perp$ **then**
      Parse $T[\mathsf{ID}]$ as (pk'_S, pk'_R)
      if $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**
        if $(\mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}) \in S \times S$ **then**
          $m^* \leftarrow \mathcal{M}$
          $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
        else
          $m^* \leftarrow m$
          $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
        $\boxed{s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m)}$
        $L[s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}] \leftarrow m$
        output $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ at $\mathtt{E}$

Interface $\mathtt{E}$ of Net

Input: $(\mathtt{inject}, s, \mathsf{ID}_s, \mathsf{ID}_r)$
  if $\mathsf{ID}_r \in J_{net}$ **then**
    $\mathtt{P}_i \leftarrow I_2^{net}[\mathsf{ID}_r]$
    Let $val_{\mathtt{P}_i} = [sk_S, pk_S, sk_R, pk_R]$
    if $T[\mathsf{ID}_s] \neq \perp$ **then**
      Parse $T[\mathsf{ID}_s]$ as (pk'_S, pk'_R)
      if $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**
        $m \leftarrow \mathsf{Unsigncrypt}(sk_R, pk'_S, s)$
        if $(\mathsf{ID}_s, \mathsf{ID}_r) \in S \times S$ **then**
          $\mathsf{bad}_1 \leftarrow (m \neq \perp \wedge L[s, \mathsf{ID}_s, \mathsf{ID}_r] = \perp)$
          $\boxed{m \leftarrow L[s, \mathsf{ID}_s, \mathsf{ID}_r]}$
        if $(\mathsf{ID}_s, \mathsf{ID}_r) \in S \times (J_{net} \setminus S)$ **then**
          $\mathsf{bad}_2 \leftarrow (m \neq \perp \wedge L[s, \mathsf{ID}_s, \mathsf{ID}_r] = \perp)$
        if $m \neq \perp$ **then**
          output (m, ID_s) at $\mathtt{P}_i$

Interface $\mathtt{E}$ of CA

Input: $(\mathtt{register}, \mathsf{ID}, val)$
  if $\mathsf{ID} \notin J$ **then**
    $T[\mathsf{ID}] \leftarrow val$
    $J \leftarrow J \cup \{\mathsf{ID}\}$
    output Success at $\mathtt{E}$
  else
    output Fail at $\mathtt{E}$

Input: `fetchAll`
  output (J, T) at $\mathtt{E}$

Interface $\mathtt{M}_i \in \mathcal{Z}$

Input: `reveal`
  output $val_{\mathtt{P}_i}$ at $\mathtt{M}_i$

Interface $\mathtt{M}_i \notin \mathcal{Z}$

Input: `reveal`
  output $\perp$ at $\mathtt{M}_i$

Figure 5.11: The second hybrid system enforces perfect confidentiality and authenticity for messages between honest parties.

Resource $\mathbf{H}_1^{\mathcal{Z}}$ and $\boxed{\mathbf{H}_2^{\mathcal{Z}}}$

Initialization

$J_{ca}, J_{net}, S \leftarrow \emptyset$
$I_1^{ca}, I_2^{ca}, I_1^{net}, I_2^{net}, T, L \leftarrow$ empty tables

Interface $\mathtt{P}_i$

Input: $(\mathtt{register}, \mathsf{ID})$
  if $\mathsf{ID}_{\mathtt{P}_i} = \perp$ **then**
    $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S$
    $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R$
    if $\mathsf{ID} \notin J_{ca}$ **then**
      $J_{ca} \leftarrow J_{ca} \cup \{\mathsf{ID}\}$
      $I_1^{ca}[\mathtt{P}_i] \leftarrow \mathsf{ID}$
      $I_2^{ca}[\mathsf{ID}] \leftarrow \mathtt{P}_i$
      $T[\mathsf{ID}] \leftarrow (pk_S, pk_R)$
      if $\mathsf{ID} \notin J_{net} \ \wedge \ I_1[\mathtt{P}_i] = \perp$ **then**
        $J_{net} \leftarrow J_{net} \cup \{\mathsf{ID}\}$
        $I_1^{net}[\mathtt{P}_i] \leftarrow \mathsf{ID}$
        $I_2^{net}[\mathsf{ID}] \leftarrow \mathtt{P}_i$
        $\mathsf{ID}_{\mathtt{P}_i} \leftarrow \mathsf{ID}$
        $val_{\mathtt{P}_i} \leftarrow [sk_S, pk_S, sk_R, pk_R]$
        if $\mathtt{M}_i \notin \mathcal{Z}$ **then**
          $S \leftarrow S \cup \{\mathsf{ID}\}$
        output Success at $\mathtt{P}_i$
      else
        output Fail at $\mathtt{P}_i$
    else
      output Fail at $\mathtt{P}_i$

Input: $(\mathtt{send}, m, \mathsf{ID})$
  if $\mathsf{ID}_{\mathtt{P}_i} \neq \perp$ **then**
    Let $val_{\mathtt{P}_i} = [sk_S, pk_S, sk_R, pk_R]$
    if $T[\mathsf{ID}] \neq \perp$ **then**
      Parse $T[\mathsf{ID}]$ as (pk'_S, pk'_R)
      if $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**
        if $(\mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}) \in S \times S$ **then**
          $m^* \twoheadleftarrow \mathcal{M}$
          $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
        else
          $m^* \leftarrow m$
          $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
        $L[s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}] \leftarrow m$
        output $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ at $\mathtt{E}$

Interface $\mathtt{E}$ **of Net**

Input: $(\mathtt{inject}, s, \mathsf{ID}_s, \mathsf{ID}_r)$
  if $\mathsf{ID}_r \in J_{net}$ **then**
    $\mathtt{P}_i \leftarrow I_2^{net}[\mathsf{ID}_r]$
    Let $val_{\mathtt{P}_i} = [sk_S, pk_S, sk_R, pk_R]$
    if $T[\mathsf{ID}_s] \neq \perp$ **then**
      Parse $T[\mathsf{ID}_s]$ as (pk'_S, pk'_R)
      if $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**
        $m \leftarrow \mathsf{Unsigncrypt}(sk_R, pk'_S, s)$
        if $(\mathsf{ID}_s, \mathsf{ID}_r) \in S \times S$ **then**
          $\mathsf{bad}_1 \leftarrow (m \neq \perp \wedge L[s, \mathsf{ID}_s, \mathsf{ID}_r] = \perp)$
          $m \leftarrow L[s, \mathsf{ID}_s, \mathsf{ID}_r]$
        if $(\mathsf{ID}_s, \mathsf{ID}_r) \in S \times (J_{net} \setminus S)$ **then**
          $\mathsf{bad}_2 \leftarrow (m \neq \perp \wedge L[s, \mathsf{ID}_s, \mathsf{ID}_r] = \perp)$
          $\boxed{m \leftarrow L[s, \mathsf{ID}_s, \mathsf{ID}_r]}$
        if $m \neq \perp$ **then**
          output (m, ID_s) at $\mathtt{P}_i$

Interface $\mathtt{E}$ **of CA**

Input: $(\mathtt{register}, \mathsf{ID}, val)$
  if $\mathsf{ID} \notin J$ **then**
    $T[\mathsf{ID}] \leftarrow val$
    $J \leftarrow J \cup \{\mathsf{ID}\}$
    output Success at $\mathtt{E}$
  else
    output Fail at $\mathtt{E}$

Input: `fetchAll`
  output (J, T) at $\mathtt{E}$

Interface $\mathtt{M}_i \in \mathcal{Z}$

Input: `reveal`
  output $val_{\mathtt{P}_i}$ at $\mathtt{M}_i$

Interface $\mathtt{M}_i \notin \mathcal{Z}$

Input: `reveal`
  output $\perp$ at $\mathtt{M}_i$

Figure 5.12: The third hybrid system enforces that no message of an honest sender can be forged.

Resource $\mathbf{H}_2^{\mathcal{Z}}$ and $\boxed{\mathbf{H}_3^{\mathcal{Z}}}$

Initialization

$J_{ca}, J_{net}, S \leftarrow \emptyset$
$I_1^{ca}, I_2^{ca}, I_1^{net}, I_2^{net}, T, L \leftarrow$ empty tables

Interface $\mathtt{P}_i$

Input: $(\mathtt{register}, \mathsf{ID})$
  if $\mathsf{ID}_{\mathtt{P}_i} = \perp$ **then**
    $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S$
    $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R$
    if $\mathsf{ID} \notin J_{ca}$ **then**
      $J_{ca} \leftarrow J_{ca} \cup \{\mathsf{ID}\}$
      $I_1^{ca}[\mathtt{P}_i] \leftarrow \mathsf{ID}$
      $I_2^{ca}[\mathsf{ID}] \leftarrow \mathtt{P}_i$
      $T[\mathsf{ID}] \leftarrow (pk_S, pk_R)$
      if $\mathsf{ID} \notin J_{net} \ \wedge \ I_1[\mathtt{P}_i] = \perp$ **then**
        $J_{net} \leftarrow J_{net} \cup \{\mathsf{ID}\}$
        $I_1^{net}[\mathtt{P}_i] \leftarrow \mathsf{ID}$
        $I_2^{net}[\mathsf{ID}] \leftarrow \mathtt{P}_i$
        $\mathsf{ID}_{\mathtt{P}_i} \leftarrow \mathsf{ID}$
        $val_{\mathtt{P}_i} \leftarrow [sk_S, pk_S, sk_R, pk_R]$
        if $\mathtt{M}_i \notin \mathcal{Z}$ **then**
          $S \leftarrow S \cup \{\mathsf{ID}\}$
        output Success at $\mathtt{P}_i$
      else
        output Fail at $\mathtt{P}_i$
    else
      output Fail at $\mathtt{P}_i$

Input: $(\mathtt{send}, m, \mathsf{ID})$
  if $\mathsf{ID}_{\mathtt{P}_i} \neq \perp$ **then**
    Let $val_{\mathtt{P}_i} = [sk_S, pk_S, sk_R, pk_R]$
    if $T[\mathsf{ID}] \neq \perp$ **then**
      Parse $T[\mathsf{ID}]$ as (pk'_S, pk'_R)
      if $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**
        if $(\mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}) \in S \times S$ **then**
          $m^* \twoheadleftarrow \mathcal{M}$
          $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
        else if $(\mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}) \in (J_{net} \times S)$ **then**
          $m^* \twoheadleftarrow \mathcal{M}$
          $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
        else
          $m^* \leftarrow m$
          $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
        $L[s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}] \leftarrow m$
        output $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ at $\mathtt{E}$

Interface $\mathtt{E}$ **of Net**

Input: $(\mathtt{inject}, s, \mathsf{ID}_s, \mathsf{ID}_r)$
  if $\mathsf{ID}_r \in J_{net}$ **then**
    $\mathtt{P}_i \leftarrow I_2^{net}[\mathsf{ID}_r]$
    Let $val_{\mathtt{P}_i} = [sk_S, pk_S, sk_R, pk_R]$
    if $T[\mathsf{ID}_s] \neq \perp$ **then**
      Parse $T[\mathsf{ID}_s]$ as (pk'_S, pk'_R)
      if $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**
        $m \leftarrow \mathsf{Unsigncrypt}(sk_R, pk'_S, s)$
        if $(\mathsf{ID}_s, \mathsf{ID}_r) \in S \times S$ **then**
          $\mathsf{bad}_1 \leftarrow (m \neq \perp \wedge L[s, \mathsf{ID}_s, \mathsf{ID}_r] = \perp)$
          $m \leftarrow L[s, \mathsf{ID}_s, \mathsf{ID}_r]$
        if $(\mathsf{ID}_s, \mathsf{ID}_r) \in S \times (J_{net} \setminus S)$ **then**
          $\mathsf{bad}_2 \leftarrow (m \neq \perp \wedge L[s, \mathsf{ID}_s, \mathsf{ID}_r] = \perp)$
          $m \leftarrow L[s, \mathsf{ID}_s, \mathsf{ID}_r]$
        if $m \neq \perp$ **then**
          output (m, ID_s) at $\mathtt{P}_i$

Interface $\mathtt{E}$ **of CA**

Input: $(\mathtt{register}, \mathsf{ID}, val)$
  if $\mathsf{ID} \notin J$ **then**
    $T[\mathsf{ID}] \leftarrow val$
    $J \leftarrow J \cup \{\mathsf{ID}\}$
    output Success at $\mathtt{E}$
  else
    output Fail at $\mathtt{E}$

Input: `fetchAll`
  output (J, T) at $\mathtt{E}$

Interface $\mathtt{M}_i \in \mathcal{Z}$

Input: `reveal`
  output $val_{\mathtt{P}_i}$ at $\mathtt{M}_i$

Interface $\mathtt{M}_i \notin \mathcal{Z}$

Input: `reveal`
  output $\perp$ at $\mathtt{M}_i$

Figure 5.13: The fourth hybrid ensures that no information about a message to an honest receiver is leaked to the adversary.

Part II

Secure Outsourced Storage

Chapter 6

A Model for Outsourced Storage

6.1 Introduction

In this chapter, we focus on security guarantees when outsourcing data on a remote storage. We initiate the study on which guarantees are needed and desired by applications. Our systematic approach does not only entail a new model for outsourced storage security, but also leads to several important insights in this branch of research as already outlined in Section 1.3.

6.1.1 Motivation

An integral and pervasive part of today's IT infrastructures are large amounts of outsourced data ranging from personal data to important enterprise backups on third-party storage providers. Depending on the various applications and sensitivity of the data, a user paying for remote storage might not fully trust in the provider's content management or security. Client-side countermeasures have to be taken into account, a prominent example of which are the protection of confidentiality and integrity of the uploaded files, or hiding the access pattern to files. A client further would like to audit the server storage to ensure that the

provider maintains all his data consistently and is not saving space by deleting a fraction of the content. That is generally known as proofs of retrievability (PoR) or provable data possession (PDP) [JK07, ABC+07]. Complementary to protocols for clients to retain security against a possibly malicious server, another line of research deals with mechanisms for secure deduplication and proofs of ownership [BKR13, HHPSP11]. These protocols allow an honest server to reduce its storage requirements while protecting against malicious clients that try to fool the server by accessing files they do not possess.

Here, our focus is on malicious server behavior. Reasons for such dishonest behavior include ordinary failures that lead to data loss or data leakage, an active break-in into the provider's infrastructure or intentional malicious server strategies. A client can employ protection mechanisms to ensure integrity, confidentiality, hide its access pattern to the data, or run regular audits to ensure that the server maintains the data reliably such that the client is able to retrieve it. Although service providers advertise availability as an important selling point, such audits are a key tool to increase the confidence or trust in the service since it is often not realistic to rely on the provider to inform reliably about an incident, either due to ignorance or due to the fear of bad reputation.

Despite sharing a common setting, previous security analyses of these tasks are often performed in different models and in a stand-alone fashion, which makes it hard to assess the overall security of a protocol (e.g. a cloud application) that involves several security schemes. We fill this gap and provide a unified composable model for capturing the security of outsourced storage. As part of this study, we justify the need for stronger security requirements from protocols than what is typically assumed in the literature. Our approach lets us develop an outsourcing scheme in modular steps that provably achieves stronger security than existing protocols.

We again develop our model in the constructive cryptography framework. Briefly, the resources we consider in this chapter are variations of so-called *server-memory resources* and the typical statements will be to construct a server-memory resource providing more from one that provides less guarantees to the client. A constructed resource can then again be used in a further construction step to achieve an even better guarantee. Looking ahead, this allows for modular protocol design and to conduct modular security analyses by dividing the a-priori complex task into several less complex steps and the security follows from the general

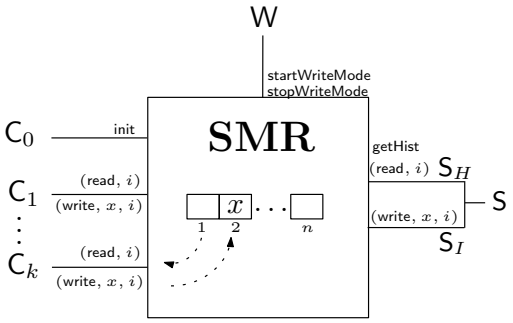


Figure 6.1: The basic server-memory resource.

composition theorem.

6.1.2 Specific Contributions

A model for untrusted storage. The basic functionality we consider is an (insecure) *server-memory resource* which we denote by **SMR** and formally specify in Section 6.2. One or several clients can write to and read from this resource via interfaces. Clients write to the memory in units of blocks, and the resource is parameterized by an alphabet Σ and the size n of blocks. The server can access the entire history of read/write requests made by the clients. To capture the active server influence on the storage, including malicious intrusion, the resource can be adaptively set into a special *server write mode* that lets the server overwrite existing data. Within the scope of this treatment here, we understand this write phase as being malicious and the server is not supposed to change any data. However, we point out that this server write mode could be used to capture intentional, honest server-side manipulations of the data storage such as in de-duplication schemes.

The decision in which "mode" the resource resides, is given directly to the environment (or distinguisher) and not to the adversary. The reason for this is important for technical and motivational reasons. Assume that the capability is provided at the malicious server interface both in the "real world" and in the "ideal world," then the simulator in the ideal world can always make use of the capability of overwriting the memory content

and nothing would prevent the simulator from doing so all the time and hence trivial protocols could be simulated. However, we want to express security guarantees in both cases, when the resource is "under attack" and when it is not. To achieve this, the "attack mode" is under the control of the environment and not the adversary. Furthermore, in certain cases we only want to give explicit security guarantees that hold only until the next attack happens (for example in the case of audits as explained later). From a motivational point of view, assigning the capability to the environment and not to attacker yields more general statements, as it also allows us to capture scenarios where the server does indeed not have the active choice to do so, but where any external event can provoke the server memory to be corrupted.

We present more secure variants of the basic server memory. In particular, in Section 6.3, we introduce the following resources:

- The *authentic* server-memory, providing authenticity of the memory content (meaning that clients detect adversarial modifications).
- The *confidential* and authentic server memory, providing secrecy (in addition to authenticity) of the memory content.
- The *secure* server memory. It provides full secrecy on the entire structure of the memory. An attacker cannot learn anything beyond the number of accesses and cannot tamper with specific logical memory cells.

We show how to construct each of these resources in Section 6.4. We then present in Section 6.5 the *auditable* versions of the above resources. An auditable server memory additionally gives the client the capability to check whether the memory has been modified or deleted, without the need to read the entire memory. We again give protocols that achieve auditable server memories in Section 6.6.

A novel notion for secure and robust outsourcing schemes. Our definition of a *secure server-memory resource* can be seen as a novel security goal: The specification demands secrecy of content and access pattern, resilience to errors, and also that active attacks cannot be targeted at specific locations. On a more technical level, our secure server-memory resource is specified as a basic server-memory resource, but where roughly

only the number of accesses leak to the server, and in particular not the content. In addition, the active influence by an attacker is restricted to being able to set a failure probability α. This parameter defines with which probability a client's read or write operation fails. This failure probability is the same for all memory locations and each memory location fails independently of other memory locations. This means that whatever the attacker does to the memory of the server, any modification will result in clients being more or less successful in reading or updating the data. In case of a failure, the client cannot read or update the corresponding block anymore. We further demand that the memory, and thus any protocol achieving it, remains operational for the faultless part of the memory and hence is *robust* in the presence of failures. As outlined above, this is technically enforced by not giving the simulator the power to always block operations and hence to abort. This makes the functionality stronger than existing models such as [AKST14, CKW13]. We give a protocol that realizes the secure server-memory resource and is an extension of the well-known Path ORAM protocol [SDS+18].

We also show that the existing definitions for access-pattern hiding and software protection are insufficient for realizing secure server-memory resources. We exemplify this by two concrete examples that do not realize a secure server memory either because the failure probability is not the same for all locations (as in [GM11]), or failures among memory locations are correlated (as in [SSS12]) and explain why this is problematic in practice.

A novel notion for audit schemes. The *auditable* server memory resources are server-memory resources with the additional client-side capability of being able to ask whether the current memory content is unchanged. This retrievability guarantee is valid if the resource is not in "adversarial write mode", as explained above, and holds up to the point when the server writes or deletes a location of the memory. A new audit has to reveal whether any change affected the client's data. Our new definition stipulates that a protocol implements a proof of retrievability if it realizes an auditiable server-memory resource from an ordinary one by possibly using additional resources like a random oracle.

To the best of our knowledge, this is the first composable security definition for audit schemes. We thereby rectify two drawbacks of existing

definitions. First, our definition and protocols guarantee that the client can download the data, if (a) the audit succeeds and (b) the adversary does not corrupt further locations after the audit. This guarantee is not required by existing definitions and not fulfilled by certain schemes such as [CKW13]. Second, existing definitions are based on the concept of knowledge-extractors: The extractor needs the client secrets and the server strategy to recover the data. As outlined in more detail in Section 6.1.5, this is not a scenario which is suitable in practice, since the client and the server would not reveal this information to each other or a third party. Our formulation does not use extractors.

For each of our server-memory resources, we show how to implement secure audits. In the particular case of secure server-memory resources, the audit reduces to a statistical estimate of the failure parameter α in combination with appropriate data replication. Our protocol resembles the protocol by Cash et al.[CKW13], but is more robust against failures: While their construction aborts when detecting an error, our scheme keeps operating even in the presence of arbitrarily many errors. As we outline in Section 6.1.3, this robustness is again needed to ensure that the data can be retrieved after a successful audit. For example, encountering just a small number of failures after an audit must not harm this guarantee.

A formal analysis of hash-based challenge-response audits. A composable formalization of storage audits in the spirit of indifferentiability and constructive cryptography [MR16, Section 7] has been envisioned in [RSS11] but has not, to the best of our knowledge, been formalized. With our formalization, we are now able to re-assess the security of the main example in [RSS11], which is the standard challenge-response audit mechanism in which the server computes a hash on the current memory content concatenated with a uniformly random challenge chosen by the client to convince the client that the data is available. We show that this scheme is not secure even in the random oracle model, contradicting the claimed security in [RSS11]. This further implies that replacing the random oracle by any provably secure iterated hash-function construction like NMAC [CDMP05, BCK96], does not have to yield secure audits. In particular, there is no contradiction to the composition theorem as claimed in [RSS11]. Note that Demay et al. [DGHM13] provide another way to

resolve this contradictory situation of [RSS11].[1] We further prove that the additional assumption needed for the hash-based audit to be secure is to restrict inputs to the random oracle to bitstrings stored in the server memory itself. This condition is sufficient for a "monolithic" random oracle with no particular underlying structure, and we show that it is in general insufficient if the random oracle is replaced by a construction (like NMAC) from ideal compression functions.

A provably secure and robust outsourcing scheme. We show how to construct an authentic server-memory resource from a basic (and thus insecure) server-memory resource in Section 6.4.1. We subsequently show how to get a confidential and authentic server-memory resource from an authentic one in Section 6.4.2, and finally achieve a secure server-memory resource from a confidential and authentic one in Section 6.4.3. Finally, in Section 6.6.2, we realize an audit mechanism as outlined above on top of the secure server-memory resources. By combining all of the above steps, the composition theorem guarantees us that the composed protocol achieves an auditable, secure server-memory resource from a basic server-memory resource (and local memory). The composed protocol is thus an efficient outsourcing scheme that provably achieves strictly stronger security than existing protocols in this realm. The protocol is resilient against any number of errors, hides the content and access pattern, does not allow targeted attacks under any circumstances, and provides an audit function.

Development of more efficient, provably secure schemes. Beyond the fact that modular steps are often simpler to analyze than an entire protocol, our modular approach has a further benefit: it allows to identify sources of inefficiency and to improve single steps in isolation without the need to re-prove the security of the overall construction: for example, developing a more efficient ORAM scheme (that in addition meets our stronger requirements) directly gives an improved overall construction. As such, we put forward this approach to provide an interface

[1] They prove that any simulator in the construction of a random oracle from ideal compression functions needs to maintain an internal state linear to the total size of all queries. This implies that the server cannot save space due to the simulator's memory consumption.

to specialized works focusing on the individual steps, such that they can directly contribute to the development of outsourcing schemes.

6.1.3 On the Importance of Composition and Robustness

Our setting has similarities with previous works that devise outsourcing schemes secure against active tampering adversaries and which build upon the foundational work by Goldreich and Ostrovsky [GO96] on software protection. There is, however, a subtle and fundamental difference between the context of outsourced storage and the context of software protection of [GO96] that seems to have gone unnoticed. In this paragraph, we show how this difference necessarily leads to strictly stronger security requirements for outsourcing schemes and even gives rise to novel security-relevant questions, which we answer in this work.

The context of [GO96] is software protection, where the goal is to prevent that an experimenter can analyze the CPU-program and learn something he could not deduce from the program specification alone. Technically, a simulator must generate an indistinguishable transcript of any experiment, solely based on the known program specification. If such a simulator exists, this means that the program effectively defeats experiments that try to figure out secret details on "how the program internally works". Following this motivation, as soon as the program encounters an error when reading a memory location, it should abort, as the error is a sign that the software is running in a tampering experiment. In the corresponding simulation, the simulator also aborts. Overall, this behavior makes perfectly sense to defeat experiments since in any honest execution, no error is expected to occur.

The context we consider is outsourcing of data and several of the above aspects do change in this realm. We present outsourcing schemes and the idealization they can achieve, like the secure server-memory resource, as a low-level primitive that exports the interface of a consistent storage with certain additional guarantees. We do not allow our primitives to abort in case an access to a location returns an error. It must stay operational for the remaining part of the memory. The decision to abort is left to the calling protocol or application that uses the memory abstraction. In our context, we want and should react to errors and not stop when detecting them. This is the first important point that makes the problem more

difficult and gives rise to the question of what level of security we can achieve in this setting. Our most secure abstraction, the secure server-memory, answers this question in a strong way: a protocol that achieves the secure server-memory not only remains operational (and efficient) when tampering is detected (a simulator cannot "abort on error" in a simulation in our model), it also makes sure that the subsequent behavior does not reveal which logical locations the client accesses, and furthermore prevents that tampering can be targeted at specific logical locations.

We illustrate two possible security issues which are overcome by using a secure outsourcing scheme (as part of a larger system) that fulfills our strongest notion.

Example 1: Information leakage due to errors. Assume that a client application stores some control information on an outsourced storage using a secure outsourcing scheme that achieves a secure server-memory. Clearly, there is no attack by which the adversary could learn when the client accesses the control information, even if the attacker knew at which logical location the control information is stored. And since the attacker can only introduce failures that are equally likely for all logical locations, the occurrence of an error during an access does not allow to infer which logical memory location was accessed. In contrast, several existing schemes based on the notion of software protection, do not guarantee this level of security and allow an attacker to obtain side-channel information about the access pattern through the observed error-pattern. This holds for example if one can approximately estimate which logical addresses are targeted by tampering with the memory, or if errors are correlated. Turned around, an observed error pattern can be a good indication on which logical locations have been accessed as further discussed in Section 6.4.4.

Example 2: Implementing secure audits. Let us focus on a protocol by Cash et al. [CKW13] that implements a proof of retrievability using a software protection scheme S that aborts on error. The following argument is independent on what security notion S fulfills exactly, the important thing is that if S aborts then the entire execution aborts. Their protocol invokes S to store the encoded data redundantly on the server, which should improve the resilience, i.e., not detecting a few errors on the server should not let the protocol fail in retrieving the data. However, since S aborts when detecting even a single (e.g., physical) error, this desired resilience practically becomes ineffective and leads to weak

guarantees: consider a very weak tampering adversary that chooses just a single, physical location on the server-memory and only tampers with this single physical location. Then, the audit is passed with high probability (the data is actually still there due to the encoding). However, the client protocol will abort before the client can actually retrieve all his data, since the error is detected beforehand, namely during a rebuild phase of S, and the execution is aborted. And as shown above in the first example, simply letting the protocol continue its actions can reveal information on the access pattern. This indicates that such a patch is actually non-trivial. In particular, if S was proven to realize a secure server-memory, then this issue is avoided. The audit protocol in Section 6.6.2 is of this type.

6.1.4 The Constructive Cryptography Setting

We consider the special case of constructive cryptography introduced in Sections 2.3.4 and 2.3.5 with one potentially dishonest interface $\mathtt{S}$ (the server), one or several honest client interfaces $\mathtt{C}_i$, and one free interface. That is, we let $\mathcal{I} = \mathcal{P} \cup \{\mathtt{S}, \mathtt{W}\}$ where $\mathcal{P} := \{\mathtt{C}_0, \ldots, \mathtt{C}_k\}$.

6.1.5 Specific Related Work

Models for outsourced storage. The security of services outsourced to untrusted third-parties has received much attention in the literature and is a prevalent topic in cloud computing. Regarding the special case of untrusted outsourced storage, Mazieres and Shasha [MS02] and later works by Androulaki et al. [ACDV14] and Cachin et al. [CSS07, CSS07, CG09, CKS09, Cac11, CDV14] formalize untrusted storage as read-write registers, where multiple clients can write to and read from the (shared) register by issuing *read and write requests*. Each request and its answer is considered an event and the view of the client is a sequence of such invocation and response events. Integrity is then defined in a property-based way as conditions on the view of the client. This model is appealing as it is expressive and the interface is simple. Our model keeps this simple structure. At each client interface a read or write request can be input. The response by the server is defined as the current value of the location (or register) which is susceptible to adversarial write operations which means that the responses may be adaptively chosen by the attacker. Our model can be seen as an universally composable variant of the above (and

where a set of cooperating clients try to obtain strong security guarantees). The benefits are stronger security guarantees and that our server-memory functionalities make the adversarial influence explicit. This allows to compare different functionalities according to their strength. It is also desirable to have a composable security definition for outsourced storage since they are naturally part of larger systems, such as distributed file systems [SvDJO12] or storage systems [SCC+10, LKMS04].

Composable notions in the realm of secure outsourced storage are unfortunately still rare. Recent examples include the works by Atteniese et al. [ADDV16], Camenisch et al.[CEM16], Liu et al. [LW16], or Apon et al.[AKST14] that illustrate the importance of filling these gaps. For example, in [ADDV16] the authors formalize the security guarantees of entangled storage in the UC framework. In order for their scheme to be secure, they work in the $\mathcal{F}_{mem}$-hybrid model, where the functionality $\mathcal{F}_{mem}$ is a memory functionality that models a server storage, where clients can upload and retrieve their values. $\mathcal{F}_{mem}$ is a quite strong assumption. For example, the adversary does not see the values uploaded to the storage. Hence, $\mathcal{F}_{mem}$ can be seen as a special case of our server memory functionality where the adversarial access is limited. Such a functionality could be obtained, for example, by defining a "wrapper" functionality along the lines of [GMPY11] that wraps an ordinary memory resource to restrict the adversarial access. This makes the assumptions on the capabilities of the attacker again explicit in order to compare different protocols.

In [CEM16], the authors investigate the security of protocols that improve the leakage resilience of imperfectly erasable memory and use the constructive cryptography framework. Imperfect erasures leak certain information to a passive adversary even after the client instructed the memory to delete the contents. This setting is fundamentally different from ours in that we are interested in security guarantees against active attacks on untrusted server storages and erasability of (local) memory is not considered. The resources in [CEM16] also make use of a free interface (also denoted world interface) to assign certain capabilities to the environment instead of the adversary to achieve general and meaningful security statements.

ORAM. Oblivious RAM is a cryptographic primitive originally introduced by Goldreich and Ostrovsky [GO96] and has become a standard approach to hiding the access pattern when accessing a cloud storage. A sequence of fundamental results have led to important security and performance improvements such as [GM11, CKW13, KLO12, DvDF+16, GGH+13, CP13, SDS+18, GHJR15, MMB15, WHC+14, FNR+15, SSS12]. Previous works define the security of ORAM schemes by requiring that different sequences of client read and write operations lead to indistinguishable sequences of accesses to the server storage. Protection against active adversaries is typically achieved by detecting malicious behavior, for example by using Merkle-Trees or authenticators [SSS12, RFY+13, AKST14], and aborting upon detection [GO96, CKW13].

PoR. The first formal security models for proofs of retrievability (PoR) were given by Juels and Kaliski [JK07] and in a similar spirit also in previous works, for example by Naor et al. [NR09] on *sublinear authentication*. The definition in [JK07] is tailored to the problem of storing static data on a server, i.e., a large file like a backup that is unlikely to be changed frequently. Roughly, an important key idea of many PoR schemes is that a redundant encoding of the file makes sure that any too small data loss is tolerable and thus need not be detected. On the other hand, by downloading some file locations at random and performing some integrity checks, a client can detect a significant amount of data loss. Subsequent publications [BJO09, DVW09, SSP13] present new and more efficient schemes as well as generalized adversarial models. In these works, the initial definition has been further carried over to the case of dynamic data. One major obstacle in the case of dynamic data is to force the server not to discard updates to (possibly a small number of) memory locations. Cash et al. [CKW13] propose a scheme based on ORAM and showed that hiding the access structure of reads and writes to the server allows for efficient PoR for dynamic data. Another solution, proposed by Chandra et al. [CKO14], shows that the problem of constructing locally decodable and locally updatable codes is strongly related to the construction of PoR schemes. Roughly, being able to update an encoding of a file F to an encoding of F' by only changing a small number of file blocks allows to reduce the problem of dynamic PoR to static PoR. Using a related guiding idea, Shi et al. [SSP13] recently proposed an (efficient) PoR

scheme for dynamic data as well. Another closely related research branch deals with models and applications for provable data possession (PDP), proposed by Ateniese et al. [ABC$^+$07, ADPMT08, ABC$^+$11]. PDP and PoR are related in spirit, but PDP is essentially a weaker definition than PoR. While the goal of proofs of retrievability is to guarantee that a file remains retrievable in full, the goal of PDP is to test if most of the file is still retrievable. Subsequent models have been proposed that deal with dynamic data, for which several protocols have been designed [ADPMT08, Kup10, EKPT09]. The security definitions of PoR (and PDP) schemes are extractor-based and are usually formalized as a game between a challenger and an adversary. A PoR scheme consists of four interactive sub-protocols executed between a stateful client and a stateful server: $\mathsf{init}(1^\kappa, \Sigma, n)$, $\mathsf{read}(i)$, $\mathsf{write}(i, v_i)$ carry their usual intended meaning, where κ is the security parameter, Σ is the alphabet and n is the size of the memory. The fourth protocol, audit, is executed to verify if the server possesses all the client's data, in which case the client returns accept.

The PoR security definition is threefold and consists of correctness, authenticity and retrievability. In this paragraph, we focus on the third security property and on the dynamic PoR game **ExtPoR** as found in [CKW13, CKO14] which we state in Figure 6.2 for completeness. For more details and variations, we refer to [JK07, SW13, CKW13, CKO14]. A scheme is said to provide retrievability, if there exists an efficient probabilistic extractor $\mathcal{E}$, such that, for every efficient server $\bar{S}$, every polynomial $p(\kappa)$ it holds that $\Pr\left[\mathbf{ExtPoR}_{\bar{S},\mathcal{E}}(\kappa, p(\kappa)) = 1\right]$ is negligible in the security parameter. Intuitively, the game formalizes that from a cheating server strategy that successfully passes an audit with good probability, it is possible to extract the correct storage content that the client uploaded. In general, the extractor is provided with the client private keys and the server strategy. The concept of a knowledge-extractor emerged from proofs of knowledge: The reasoning in proofs of knowledge is that if a dishonest prover passes the test with good probability using some arbitrary strategy, then this strategy could be used by the prover himself to effectively calculate the witness by virtue of the extractor algorithm, which is an algorithm that extracts the witness by executing and rewinding the prover's strategy.

Although the extractor-based approach to PoR includes a meaningful thought-experiment, it has a major drawback concerning client-side se-

Retrievability Experiment $\mathbf{ExtPoR}_{\bar{S},\mathcal{E}}(\kappa; p)$

1.) $\bar{S}$ outputs a valid protocol sequence $P := (op_0, op_1, \ldots, op_q)$, i.e., a sequence of invocations where $op_0 = \mathsf{init}(1^\kappa, \Sigma, n)$ and, for $j > 0$, $op_j \in \{\mathsf{read}(i), \mathsf{write}(i, v_i), \mathsf{audit}\}$ with $i \in [n]$ and $v_i \in \Sigma$. We denote by M the correct memory content after an ideal execution of the sequence P.

2.) The challenger creates an instance of an honest client $\mathcal{C}$ and executes the whole sequence P between $\mathcal{C}$ and $\bar{S}$. After the execution, let $\mathcal{C}_{\text{fin}}$ and $\bar{S}_{\text{fin}}$ denote the state of the client and server (including its random coins), respectively. We define the following probability over the random coins of the client during the audit:

$$\mathsf{Succ}\left(\bar{S}_{\text{fin}}\right) := \Pr\left[\bar{S}_{\text{fin}} \overset{\mathsf{audit}}{\longleftrightarrow} \mathcal{C}_{\text{fin}} \Rightarrow \mathsf{accept}\right]$$

3.) Run $M' \leftarrow \mathcal{E}^{\bar{S}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^n, 1^p)$, where the extractor gets black-box rewinding access to the server strategy and in addition a description of the client strategy.

4.) If $\mathsf{Succ}\left(\bar{S}_{\text{fin}}\right) \geq \frac{1}{p}$ but $M' \neq M$, then output 1 (the server wins the game). Otherwise, output 0.

Figure 6.2: The retrievability experiment between a (malicious) server $\bar{S}$, an extractor $\mathcal{E}$, and the challenger.

curity guarantees: If an audit is successful, the availability of the data, which is of major concern to the client, is only guaranteed through the execution of the extractor, which needs to access the server strategy and the secret state of the client. Both parties are unlikely to disclose this sensitive information. No server would reveal its entire state and no client would reveal its secret keys.

6.2 Basic Server-Memory Resource

Our basic server-memory resource allows clients to read and write data blocks, where each block is encoded as an element v of some alphabet Σ (a finite non-empty set). An element of Σ is considered a data block. At the server interface, denoted $\mathtt{S}$, the resource provides the entire history of accesses made by the clients (modeling the information leakage via a server log file in practice), and allows the server to overwrite existing

data blocks. To syntactically separate the former capability (modeling data leakage), from the latter, capability (modeling active influence), we formally divide interface $\mathtt{S}$ into two sub-interfaces which we denote by $\mathtt{S}_H$ (for honest but curious) and $\mathtt{S}_I$ (for intrusion). The server can only overwrite data blocks if the resource is set into a special write mode. The distinguisher (or environment) is given the capability to adaptively enable and disable this write mode at the free interface $\mathtt{W}$. The combination of capabilities at interfaces $\mathtt{W}$ and $\mathtt{S}_I$ allows our model to capture different types of adversarial influence, including adaptively setting return values of client read operations, or to model phases in which no server write access is possible at all. We present the basic server-memory resource, called $\mathbf{SMR}^{k}_{\Sigma,n}$, in detail in Figure 6.3.

Our formalization is more general than the simple client-server setting in that it takes into account several clients that access the resource, each via their interface $\mathtt{C}_i$. The parameters of the resource are the number of clients k, the alphabet Σ, and the number of blocks. The interface $\mathtt{C}_0$ is the initialization interface and is used to set up the initial state of the resource (for example as a first step in the protocol). Only after the resource is initialized, indicated by the input `initComplete` at $\mathtt{C}_0$, the client interfaces become active and can update the state. We assume that (adversarial) server write operations only happen after the initialization is complete. Interface $\mathtt{C}_0$ can be thought of as being assigned to a special party or simply to a dedicated client whose first actions are to initialize the resource. Also note that for the sake of simplicity we only specify client requests that occur when the server-write mode is turned off. The complementary case simply follows along the lines of [AKST14]: In case of a currently active intrusion, we would just let the adversary define the return value (i.e., define the cell content before the answer to the client is generated). All our protocols are easily seen to be secure with respect to this model: if a client access to a cell happened during an active intrusion, then this can be mapped to our model by (1) inserting an additional active intrusion step before the client access in which the distinguisher can define all return values (via interface $\mathtt{S}_I$), (2) de-activating the server write mode and execute the client access, (3) activate the server write mode again.

The basic server-memory resource constitutes the core element of our model and serves as the fundamental building block for numerous applications in the realm of cloud storage as discussed in the previous

sections. In this thesis, we elaborate along the lines of securing the memory resource against a malicious server. Other possible directions could include the formalization of distributed file-systems, access control mechanisms, or entangled storage which we do not consider here.

6.3 More Secure Memories

In this section, we present server-memory resources that offer more security guarantees for the clients in that they restrict the capabilities of the server.

Authentic server-memory resource. An authentic server-memory resource enhances the basic server-memory resource by restricting the capabilities at the active interface $\mathtt{S}_I$. Instead of being capable to modify existing data blocks, the server can either delete data blocks, via input $(\mathtt{delete}, i)$ at $\mathtt{S}_I$, or restore previously deleted data blocks, via input $(\mathtt{restore}, i)$ at $\mathtt{S}_I$. A deleted data block is indicated by the special symbol ϵ. A client accessing the location of a deleted data block simply receives ϵ as an answer. We formally describe the authentic server-memory resource $\mathbf{aSMR}^k_{\Sigma,n}$ in Figure 6.4.

Confidential server-memory resource. The confidential and authentic server-memory resource, denoted $\mathbf{cSMR}^k_{\Sigma,n}$, is formally specified in Figure 6.5. It enhances the authentic server-memory resource by restricting the access at the server interface $\mathtt{S}_H$ in that each server read operation simply returns $\lambda \in \Sigma$. Furthermore, the history of client accesses only reveal the location, but not the value that was read or written.

Secure (oblivious) server-memory resource. We present the secure (and oblivious) server-memory resource in Figure 6.6. This resource offers the strongest guarantees for the clients. First, the access pattern does not leak to the server apart from the number of accesses made. Second, the adversarial influence is now limited to setting a corruption or "pollution" parameter α. On each client read or write operation $(\mathtt{read}, i)$ or $(\mathtt{write}, i, x)$ the operation fails with probability α and the cell i is considered deleted. This expresses the inability of an intruder to mount a targeted attack on chosen blocks. His influence pollutes the entire memory in the specific way of increasing (or decreasing) the probability

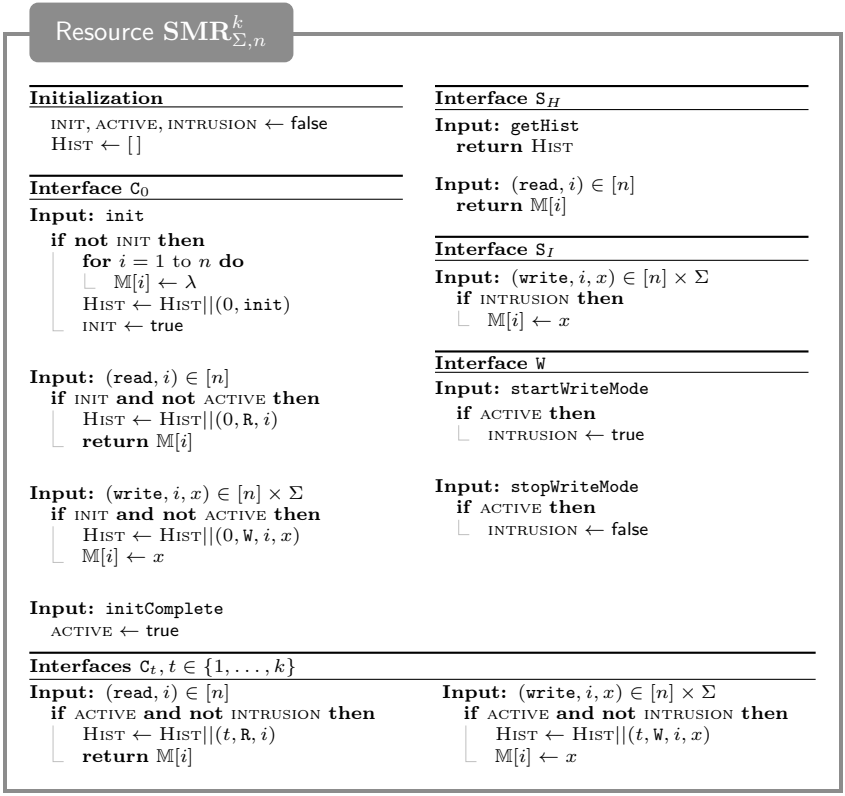


Figure 6.3: Description of the insecure server-memory resource.

Resource $\mathbf{aSMR}^{k}_{\Sigma,n}$

The definition of **aSMR** is identical to **SMR** *except* for the adversarial influence at interface $\mathtt{S}_I$ and the reaction on writing to a corrupted memory location:

Interfaces $\mathtt{C}_t, t \in \{1, \ldots, k\}$

Input: $(\mathtt{read}, i) \in [n]$
  if ACTIVE **and not** INTRUSION **then**
    HIST $\leftarrow$ HIST$||(t, \mathtt{R}, i)$
    return $\mathbb{M}[i]$

Input: $(\mathtt{write}, i, x) \in [n] \times \Sigma$
  if ACTIVE **and not** INTRUSION **then**
    if $\mathbb{M}[i] \neq \epsilon$ **then**
      HIST $\leftarrow$ HIST$||(t, \mathtt{W}, i, x)$
      $\mathbb{M}[i] \leftarrow x$
    else
      HIST $\leftarrow$ HIST$||(t, \mathtt{Fail}, i, x)$
      return ϵ

Interface $\mathtt{S}_I$

Input: $(\mathtt{delete}, i) \in [n]$
  if INTRUSION **then**
    $\mathbb{M}[i] \leftarrow \epsilon$

Input: $(\mathtt{restore}, i) \in [n]$
  if INTRUSION **then**
    if $\exists k, x, t :$ HIST$[k] = (t, \mathtt{W}, i, x)$ **then**
      $k_0 \leftarrow \max\{k \mid \exists t, x :$ HIST$[k] = (t, \mathtt{W}, i, x)\}$
      Parse HIST$[k_0]$ as $(t, \mathtt{W}, i, x_0)$
      $\mathbb{M}[i] \leftarrow x_0$
    else
      $\mathbb{M}[i] \leftarrow \lambda$

Figure 6.4: The authentic server-memory resource (only differences to **SMR** shown).

of a failure. In particular, our ideal functionality demands that each cell or block fails independently and with the same probability (if it had not failed before). Our formulation of this resource, which we denote by $\mathbf{sSMR}^{k,t_{\mathrm{rep}}}_{\Sigma,n}$ and describe in Figure 6.6, is slightly more general than just described: it is parameterized as before by the number of clients k, the alphabet Σ, the size n, and additionally by a tolerance t_{rep} (considered as the replication factor) that formalizes the resilience against failures. Intuitively, only after t_{rep} read or write operations for location i have failed, i is considered as deleted, which of course includes the standard case $t_{\mathrm{rep}} = 1$. This guarantee, although quite strong, seems appealing in practice and is realizable as we prove in the next section.[2]

[2]One could again complement the specification with the behavior upon a client request during an active intrusion phase. As in [AKST14], one could let the adversary decide on the success or failure of a client request in that case. More technically,

Resource $\mathbf{cSMR}^{k}_{\Sigma,n}$

The definition of **cSMR** is identical to **SMR** *except* for the information the server and the adversary learn about the stored data:

Interface $\mathtt{S}_H$

Input: `getHist`
  $\textsc{Hist}' \leftarrow [\,]$
  for $j = 1$ to $|\textsc{Hist}|$ **do**
    $q \leftarrow \textsc{Hist}[j]$
    if $q = (t, \mathtt{W}, i, x)$ for some t, x, i **then**
      $\textsc{Hist}' \leftarrow \textsc{Hist}' || (t, \mathtt{W}, i)$
    if $q = (t, \mathtt{Fail}, i, x)$ for some t, x, i **then**
      $\textsc{Hist}' \leftarrow \textsc{Hist}' || (t, \mathtt{Fail}, i)$
    if $q = (t, \mathtt{R}, i)$ for some t, i **then**
      $\textsc{Hist}' \leftarrow \textsc{Hist}' || (t, \mathtt{R}, i)$
  return $\textsc{Hist}'$

Input: $(\mathtt{read}, i) \in [n]$
  return λ

Interface $\mathtt{S}_I$

Input: $(\mathtt{delete}, i) \in [n]$
  if INTRUSION **then**
    $\mathbb{M}[i] \leftarrow \epsilon$

Input: $(\mathtt{restore}, i) \in [n]$
  if INTRUSION **then**
    if $\exists k, x, t : \textsc{Hist}[k] = (t, \mathtt{W}, i, x)$ **then**
      $k_0 \leftarrow \max\{k \mid \exists t, x : \textsc{Hist}[k] = (t, \mathtt{W}, i, x)\}$
      Parse $\textsc{Hist}[k_0]$ as $(t, \mathtt{W}, i, x_0)$
      $\mathbb{M}[i] \leftarrow x_0$
    else
      $\mathbb{M}[i] \leftarrow \lambda$

Figure 6.5: The authentic and confidential server-memory resource (only differences to **SMR** shown).

It further seems to be a desriable abstraction on its own, for example in the context of data replication where the assumption that blocks fail independently is crucial. It further allows for straightforward statistical predictions of this error parameter. One could imagine to weaken this resource by considering correlations among failures, or to allow different cells to fail with different probabilities. We only consider the strongest variant and show how to achieve it.

6.4 Constructions

In this section, we show how to construct stronger server-memory resources from weaker ones. For each construction, we need to specify the protocol for the clients by means of a converter which every client attaches to

following the general approach to map such cases into our model, we would simply allow the adversary to adjust the corruption parameter α in Figure 6.6 during such a request before calling the sampling function. The security of the protocols is not be affected by such a change.

Resource $\mathbf{sSMR}^{k,t_{\mathsf{rep}}}_{\Sigma,n}$

Initialization

INIT, ACTIVE, INTRUSION $\leftarrow$ false
$\alpha \leftarrow 0$; HIST $\leftarrow [\,]$
$c_i \leftarrow 1$ for all $i \in [\ell]$

Interface $\mathtt{C}_0$

Input: `init`
  if not INIT **then**
    for $i = 1$ to n **do**
      for $j = 1$ to t_{rep} **do**
        $\mathbb{M}[i,j] \leftarrow \lambda$
    HIST $\leftarrow$ HIST$||(0, \mathtt{init})$
    INIT $\leftarrow$ true

Input: $(\mathtt{read}, i) \in [n]$
  if INIT **and not** ACTIVE **then**
    HIST $\leftarrow$ HIST$||(0, \mathtt{R}, i)$
    return $\mathbb{M}[i, 1]$

Input: $(\mathtt{write}, i, x) \in [n] \times \Sigma$
  if INIT **and not** ACTIVE **then**
    for $j = 1$ to t_{rep} **do**
      HIST $\leftarrow$ HIST$||(0, \mathtt{W}, i, x)$
      $\mathbb{M}[i,j] \leftarrow x$

Input: `initComplete`
  ACTIVE $\leftarrow$ true

Interface $\mathtt{S}_H$

Input: `getHist`
  HIST$'$ $\leftarrow [\,]$
  for $j = 1$ to $|$HIST$|$ **do**
    $q \leftarrow$ HIST$[j]$
    if $q \in \{(t, \mathtt{W}, i, x), (t, \mathtt{R}, i)\}$ **then**
      HIST$'$ $\leftarrow$ HIST$'||(t, \mathtt{Access})$
    else
      HIST$'$ $\leftarrow$ HIST$'||q$
  return HIST$'$

Input: $(\mathtt{read}, i) \in [n]$
  return λ

Interface $\mathtt{S}_I$

Input: $(\mathtt{pollute}, \rho) \in [0, 1]$
  if INTRUSION **then**
    $\alpha \leftarrow \rho$

Input: $(\mathtt{reducePollution}, \delta) \in [0, \alpha]$
  if INTRUSION **then**
    $\alpha \leftarrow \alpha - \delta$

Interface `W`

Input: `startWriteMode`
  if ACTIVE **then**
    INTRUSION $\leftarrow$ true

Input: `stopWriteMode`
  if ACTIVE **then**
    INTRUSION $\leftarrow$ false

Interfaces $\mathtt{C}_t, t \in \{1, \ldots, k\}$

Input: $(\mathtt{read}, i) \in [n]$
  if ACTIVE **and not** INTRUSION **then**
    if $\mathbb{M}[i, c_i] \neq \epsilon$ **then**
      $Z \leftarrow$ Bernoulli(α)
      if $Z = 0$ **then**
        HIST $\leftarrow$ HIST$||(t, \mathtt{R}, i)$
        return $\mathbb{M}[i, c_i]$
      else
        HIST $\leftarrow$ HIST$||(t, \mathtt{Failed})$
        $\mathbb{M}[i, c_i] \leftarrow \epsilon$
        if $c_i < t_{\text{rep}}$ **then**
          $c_i \leftarrow c_i + 1$
        return ϵ
    else
      $Z \leftarrow$ Bernoulli(α)
      if $Z = 0$ **then**
        HIST $\leftarrow$ HIST$||(t, \mathtt{Access})$
      else
        HIST $\leftarrow$ HIST$||(t, \mathtt{Failed})$
      if $c_i < t_{\text{rep}}$ **then**
        $c_i \leftarrow c_i + 1$
      return ϵ

Input: $(\mathtt{write}, i, x) \in [n] \times \Sigma$
  if ACTIVE **and not** INTRUSION **then**
    RET$_j \leftarrow$ ok for $j = 1 \ldots t_{\text{rep}}$
    for $j = 1$ to t_{rep} **do**
      if $\mathbb{M}[i,j] \neq \epsilon$ **then**
        $Z \leftarrow$ Bernoulli(α)
        if $Z = 0$ **then**
          HIST $\leftarrow$ HIST$||(t, \mathtt{W}, i, x)$
          $\mathbb{M}[i,j] \leftarrow x$
        else
          HIST $\leftarrow$ HIST$||(t, \mathtt{Failed})$
          $\mathbb{M}[i,j] \leftarrow \epsilon$
          RET$_j \leftarrow \epsilon$
      else
        RET$_j \leftarrow \epsilon$
        $Z \leftarrow$ Bernoulli(α)
        if $Z = 0$ **then**
          HIST $\leftarrow$ HIST$||(t, \mathtt{Access})$
        else
          HIST $\leftarrow$ HIST$||(t, \mathtt{Failed})$
    return $(\text{RET}_1, \ldots, \text{RET}_{t_{\text{rep}}})$

Figure 6.6: Description of the secure server-memory resource.

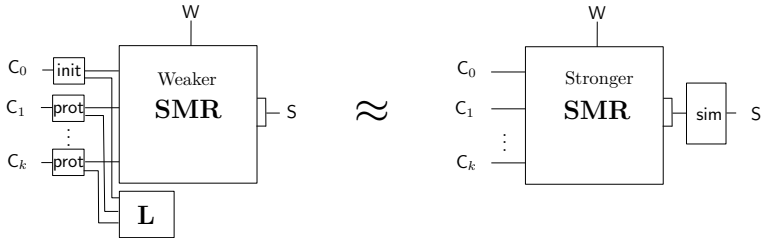


Figure 6.7: Illustration of the security condition of constructions among server-memory resources.

its interface. We further have to provide a converter that describes the initialization step (generating cryptographic keys etc.) and which is attached at interface $\mathtt{C}_0$. To show that a protocol achieves a construction, we prove both conditions stated in Definition 2.3.3. More specifically to this section, the default behavior of the potentially dishonest server interface is specified by the dummy converter honSrv that does not answer any query at its outer interface and does not give any input to the server-memory resource.

The protocols we present make use of a local memory $\mathbf{L}$ shared among all clients. At each interface $\mathtt{C}_i$ of $\mathbf{L}$, the usual read and write capabilities are available. The server does not have access to this resource. An illustration is shown in Figure 6.7. Note that client accesses to the resources are sequential in our model (which is trivially the case in a single client setting or achievable via Dekker's or Peterson's mutual-exclusion algorithms using the shared memory $\mathbf{L}$).

6.4.1 Authentic Server-Memory from Basic Server-Memory Resources

Following Blum et al. [BEG+94], we build a tree structure on top of the outsourced data blocks to protect their authenticity (and freshness). Assume the size of the memory is ℓ, then the tree is a binary search tree with ℓ leaves, where each leaf corresponds to a data block. For simplicity, and without loss of generality, we assume that ℓ is a power of 2. In [BEG+94], each leaf is associated with a timestamp indicating the number of times the block was updated. The timestamp of an internal node is

defined as the sum of the timestamps of its two children. We refer to this condition on the timestamps as the *tree invariant*. The timestamp of the root of the tree corresponds to total number of times the client has accessed the server-memory resource and is stored in a reliable local memory.

Protocol and notation. We denote the full binary tree for a memory of size ℓ as $T^{(\ell)}$. The tree has $2\ell - 1$ nodes which are mapped to the linear storage of **SMR** (of size $2\ell - 1$) and a local reliable memory **L** as follows[3]: The leaf node at location $\ell + i - 1$ of **SMR** stores the value x_i of the (logical) memory. Internal nodes only contain a timestamp (and an authentication tag) and are stored in **SMR**. We denote the node at location r of **SMR** by N_r (for $r > 0$) and denote the root as N_0. For a node N_r, we denote by t_r its timestamp, and for a leaf node we additionally denote by $x_i \in \Sigma$ its associated data block ($i = r - \ell + 1$). To bind the contents of a node to its actual location on the server, we make use of a MAC function $f_{sk}(\cdot)$. The root node N_0 is not stored in **SMR** but on a reliable, local memory **L**. In summary, the format of the nodes are as follows:

$$N_r = \begin{cases} (x_i, t_r, f_{sk}(r, x_i, t_r)) & \textbf{if } r \geq \ell \quad (i = r - \ell + 1) \\ (t_r, f_{sk}(r, t_r)) & \textbf{if } r < \ell \\ t_0 & \textbf{if } r = 0. \end{cases}$$

To read a value x_i of the (logical) memory, the client retrieves all nodes on the path from the root to the leaf node $N_{\ell+i-1}$ and all their children. This is sometimes denoted to as the siblings path from the root to that leaf. We denote this sub-tree (consisting of $2 \log \ell$) nodes by $T_{\ell+i-1}$ to make the index i appear explicitly. On each access to the logical data block i, the client verifies all authentication tags and checks the invariant in $T_{\ell+i-1}$, i.e., that for each node N_r, the timestamp is the sum of its children's timestamps, i.e., that $t_r = t_{2r+1} + t_{2r+2}$. If all checks succeed the tree $T_{\ell+i-1}$ is said to be *valid*. To write a new value to a leaf node $N_{\ell+i-1}$, one first retrieves $T_{\ell+i-1}$ and verifies that it is

[3]Note that the number of storage locations of **SMR** is about two times as large as the logical locations we want to protect. This, however, does not imply a storage overhead of a factor of two in practice, since the information stored in internal nodes is only a timestamp and not an entire data block.

Converter $\mathsf{init}_{\mathrm{auth}}$

Interface `out`

Input: `init` :
  $sk \twoheadleftarrow \mathcal{K}$; $N_0 \leftarrow 0$;
  output $(\texttt{write}, 1, N_0)$ at `in` to **L**
  output $(\texttt{write}, 2, sk)$ at `in` to **L**
  output `init` at `in` to **SMR**
  for $i = 2$ to $\ell - 1$ **do**
    $N_i \leftarrow (0, f_{sk}(i, 0))$
    output $(\texttt{write}, i, N_i)$ at `in` to **SMR**
  for $i = \ell$ to $2\ell - 1$ **do**
    $N_i \leftarrow (\lambda, 0, f_{sk}(i, \lambda, 0))$
    output $(\texttt{write}, i, N_i)$ at `in` to **SMR**

Input: $(\texttt{read}, i)$ or $(\texttt{write}, i, x)$:
  Defined the same as for $\mathsf{auth}_{\mathrm{RW}}$ in Figure 6.9

Input: `initComplete` :
  output `initComplete` at `in` to **SMR**

Figure 6.8: The initialization protocol for the realization of an authentic server memory.

valid. Then, one updates the value of the leaf, its timestamp and the authentication tag and subsequently updates the timestamps and the authentication tags of all nodes on the path to the root to restore the invariant of tree $T_{\ell+i-1}$. Finally, all nodes are written back to their original location. It is straightforward to cast this protocol as a converter for the clients, denoted by $\mathsf{auth}_{\mathrm{RW}}$ and specified in Figure 6.9. The generation of cryptographic keys and the initial setup of the tree are formally specified in the initialization converter $\mathsf{init}_{\mathrm{auth}}$ found in Figure 6.8.

Intuitively, this protocol is secure since no adversary can inject a new value at any memory location, as each node is bound to a memory location on the server. Additionally, replaying an older value is not possible as an older value has a smaller timestamp and if the client verifies the tree's invariant, its reliably stored value N_0 is too large. Formally, we prove the following theorem.

Theorem 6.4.1. *Let $k, \ell \in \mathbb{N}$ and let $\Sigma_1 = \Sigma \times Z_q \times \mathcal{T}$ for some alphabet Σ. The protocol $\mathsf{auth} := (\mathsf{init}_{\mathrm{auth}}, \mathsf{auth}_{\mathrm{RW}}, \ldots, \mathsf{auth}_{\mathrm{RW}})$ (with k copies of $\mathsf{auth}_{\mathrm{RW}}$) described above based on a MAC function f with tag*

Converter $\mathsf{auth}_{\mathrm{RW}}$

Interface `out`

Input: $(\texttt{read}, i)$
  $T \leftarrow$ GetValidSubtree$(\ell + i - 1)$
  if $T \neq \emptyset$ **then**
    Parse node $N_{\ell+i-1}$ (of tree T) as $(x, t, f_{sk}(\ell + i - 1, x, t))$
    return x
  else
    return ϵ

Input: $(\texttt{write}, i, x)$
  $T \leftarrow$ GetValidSubtree$(\ell + i - 1)$
  if $T \neq \emptyset$ **then**
    $N_{\ell+i-1} \leftarrow (x, t_{\ell+i-1} + 1, f_{sk}(\ell + i - 1, x, t_{\ell+i-1} + 1))$
    for each internal node $N_i \in T$ **do**
      $N_i \leftarrow (t_i + 1, f_{sk}(i, t_i + 1))$
    for each node $N_i \in T$ with $i > 0$ **do**
      output $(\texttt{write}, i, N_i)$ at `in` to **SMR**
    $N_0 \leftarrow N_0 + 1$
    output $(\texttt{write}, 1, N_0)$ at `in` to **L**
  else
    return ϵ

---

function GetValidSubtree(i)
  Retrieve $T_{\ell+i-1}$, i.e., all nodes N_r on the path from leaf node $N_{\ell+i-1}$ to the root N_0 (and their siblings) in descending order of index via:
    -**output** $(\texttt{read}, 1)$ at `in` to **L** to read N_0
    -**output** $(\texttt{read}, r)$ at `in` to **SMR** to read N_r
  for each node $N_r \in T_{\ell+i-1}$ with $r > 0$ **do**
    Case $N_r = (t_r, tag)$**:**
      if $f_{sk}(r, t_r) \neq tag$ **then**
        return $\emptyset$
    Case $N_r = (x, t_r, tag)$**:**
      if $f_{sk}(r, x, t_r) \neq tag$ **then**
        return $\emptyset$
  for each node $N_r \in T_{\ell+i-1}$ with $r > 0$ **do**
    par $\leftarrow \lceil \frac{i}{2} \rceil - 1$ ▷ index of parent
    $\mathrm{ch}_1 \leftarrow 2 \cdot \mathrm{par} + 1$ ▷ index of 1st child
    $\mathrm{ch}_2 \leftarrow 2 \cdot \mathrm{par} + 2$ ▷ index of 2nd child
    if $t_{\mathrm{par}} \neq t_{\mathrm{ch1}} + t_{\mathrm{ch2}}$ **then**
      return $\emptyset$
  return $T_{\ell+i-1}$

Figure 6.9: The converter for the clients to realize an authentic server memory from a basic server memory.

space $\mathcal{T}$ constructs the authentic server memory $\mathbf{aSMR}^{k}_{\Sigma,\ell}$ from the basic server memory $\mathbf{SMR}^{k}_{\Sigma_1,2\ell}$ and a local memory $\mathbf{L}$ (of constant size), with respect to the simulator $\mathsf{sim}_{\mathrm{auth}}$ as defined in Figure 6.10 and the pair $(\mathsf{honSrv}, \mathsf{honSrv})$. More specifically, we design a reduction ρ such that for all distinguishers $\mathbf{D}$ and their associated adversaries $\mathcal{A} := \rho(\mathbf{D})$,

$$\Delta^{\mathbf{D}}(\mathsf{honSrv}^{\mathsf{S}}\ \mathsf{auth}_{\mathcal{P}}[\mathbf{L}, \mathbf{SMR}^{k}_{\Sigma_1,2\ell}], \mathsf{honSrv}^{\mathsf{S}}\mathbf{aSMR}^{k}_{\Sigma,\ell}) = 0$$

and $$\Delta^{\mathbf{D}}(\mathsf{auth}_{\mathcal{P}}[\mathbf{L}, \mathbf{SMR}^{k}_{\Sigma_1,2\ell}], \mathsf{sim}^{\mathsf{S}}_{\mathrm{auth}}\mathbf{aSMR}^{k}_{\Sigma,\ell}) \leq \mathsf{Adv}^{\mathit{eu\text{-}cma}}_{f,\mathcal{A}}.$$

Proof. The correctness condition is obvious and we only give a proof of the security condition. We analyze the input-output behavior of both systems involved. To this end, we consider the possible inputs at each interface.

On input `init`, `initComplete` at interface $\mathtt{C}_0$: Upon the `init`-query, the protocol $\mathsf{init}_{\mathrm{auth}}$ of the real system $\mathsf{auth}_{\mathcal{P}}[\mathbf{L}, \mathbf{SMR}^{k}_{\Sigma_1,2\ell}]$ generates a secret MAC key sk and initializes the basic memory resource. Subsequently, the protocol writes the nodes of $T^{(\ell)}$ to the server memory, except for the root N_0 which is stored in the local memory. This initialization adds $2\ell - 2$ entries to the history HIST of **SMR**. After this initialization phase, HIST reads $(0, \mathtt{init})||(0, \mathtt{W}, 1, N_1)||\ldots||(0, \mathtt{W}, 2\ell - 1, N_{2\ell-1})$. Any subsequent `read` query will return the fixed value $\lambda \in \Sigma$.

In the ideal system $\mathsf{sim}^{\mathsf{S}}_{\mathrm{auth}}\mathbf{aSMR}^{k}_{\Sigma,\ell}$, the query initializes the memory to the fixed value $\lambda \in \Sigma$ and adds the entry $(0, \mathtt{init})$ to HIST. Since this is the first entry of HIST, the simulator will replace this entry by its locally simulated list L_{init} that consists of the $2\ell - 2$ entries as above (where the key MAC key sk is chosen locally by the simulator).

Finally, on input `initComplete`, both systems deactivate interface $\mathtt{C}_0$ and the other client interfaces are operational from this point onwards.

On input $(\mathtt{read}, i)$ at interface $\mathtt{C}_k$: On this query, both protocols, that is $\mathsf{init}_{\mathrm{auth}}$ (in case $k = 0$) and $\mathsf{auth}_{\mathrm{RW}}$ (in case $k > 0$), retrieve all the nodes of the sub-tree $T_{\ell+i-1}$. This sub-tree contains the leaf node $N_{\ell+i-1} = (x_i, t, f_{sk}((\ell + i - 1, t, x_i)))$ of $T(\ell)$ which stores the

Converter $\mathsf{sim}_{\text{auth}}$

Initialization

$sk \twoheadleftarrow \mathcal{K}$
Initialize $2\ell - 1$ nodes N_i as in Figure 6.8
$L_{init} \leftarrow (0, \texttt{init})||(0, \texttt{W}, 1, N_1)|| \ldots ||(0, \texttt{W}, 2\ell - 1, N_{2\ell-1})$
$pos \leftarrow 1$

Interface $\mathtt{S}_H$

Input: `getHist` :
UpdateLog
return L

Input: $(\texttt{read}, r) \in [2\ell - 1]$:
UpdateLog
return N_r

Interface $\mathtt{S}_I$ (INTRUSION = true)

Input: $(\texttt{write}, r, x) \in [2\ell - 1] \times (\Sigma \times Z_n \times \mathcal{T})$
UpdateLog
Determine the last entry in L that wrote value N in to location i
if $N \neq x$ **then**
  $N_r \leftarrow x$
  for each leaf node $N_{\ell+i-1}$ whose sub-tree $T_{\ell+i-1}$ is not valid **do**
    output $(\texttt{delete}, i)$ at `in` to **aSMR**
else
  for each leaf node $N_{\ell+i-1}$ whose sub-tree $T_{\ell+i-1}$ is valid **do**
    output $(\texttt{restore}, i)$ at `in` to **aSMR**
  $N_r \leftarrow N$

procedure UpdateLog
  output `getHist` at `in` to **aSMR**
  Let Hist be the returned value
  for $j = pos$ to $|\text{Hist}|$ **do**
    if $\text{Hist}[j] = (0, \texttt{init})$ **then**
      $L \leftarrow L_{init}$
    else if $\text{Hist}[j] = (k, \texttt{R}, i)$ **then**
      Let $T_{\ell+i-1}$ be the sub-tree of simulated leaf node $N_{\ell+i-1}$
      for each node $N_r \in T_{\ell+i-1}$ in decreasing order of index **do**
        $L \leftarrow L||(k, \texttt{R}, r)$
    else if $\text{Hist}[j] = (k, \texttt{W}, i, x)$ **then** ▷ Successful write operation
      Let $T_{\ell+i-1}$ be the (valid) sub-tree of simulated leaf node $N_{\ell+i-1}$
      for each node $N_r \in T_{\ell+i-1}$ in decreasing order of index **do**
        $L \leftarrow L||(k, \texttt{R}, r)$
      for each node $N_r \in T_{\ell+i-1}$ in decreasing order of index **do**
        Update N_r according to the protocol (Figure 6.9)
        $L \leftarrow L||(k, \texttt{W}, r, N_r)$
    else if $\text{Hist}[j] = (k, \texttt{Fail}, i, x)$ **then** ▷ Failed write operation
      Let $T_{\ell+i-1}$ be the sub-tree of simulated leaf node $N_{\ell+i-1}$
      for each node $N_r \in T_{\ell+i-1}$ in decreasing order of index **do**
        $L \leftarrow L||(k, \texttt{R}, r)$
  $pos \leftarrow |\text{Hist}| + 1$

Figure 6.10: The simulator for the construction of an authenticated memory.

value of memory location i. Furthermore, $T_{\ell+i-1}$ consists of all nodes on the path from that leaf to the root together with their children. To retrieve this sub-tree, the protocol issues $2\log\ell - 1$ `read`-queries. The history HIST hence is increased by the list of $2\log\ell - 1$ value $(k, \mathtt{R}, r_1)||...||(k, \mathtt{R}, r_{2\ell-1})$, where the indices r_j are ordered in decreasing order according to their location in **SMR**. Afterwards, the protocol checks the validity of each authentication tag and checks the tree's invariant (i.e., that the sum of the children's timestamps is equal to the parent's timestamp). If all checks succeed, x_i is output (which is the last value written to this location), and otherwise ϵ is output.

In system $\mathsf{sim}^{\mathtt{S}}_{\text{auth}}\mathbf{aSMR}^{k}_{\Sigma,\ell}$, system **aSMR** answers the query with the current memory content of cell i. If the cell is not corrupted, the last value written is output at interface $\mathtt{C}_k$. If the cell is corrupted, ϵ is output. In order for the simulator $\mathsf{sim}_{\text{auth}}$ to emulate this view, it internally simulates the tree T^{ℓ} and, after each adversarial query at interfaces $\mathtt{S}_I$ and $\mathtt{S}_H$ keeps track of which sub-trees $T_{\ell+r-1}$ are valid, and if not, corrupts the cell r of **aSMR** (cf. behavior at interface $\mathtt{S}_H$ and $\mathtt{S}_I$ below). This enforces the consistency between successful reads and valid subtrees exactly as in the real system. Additionally, the next time the simulator is activated, it will update its simulated history L accordingly: if this read-request $(k, \mathtt{R}, i)$ is the qth entry in HIST of **aSMR**, then, in procedure UPDATELOG, this qth entry will lead to the increase of $2\log\ell - 1$ `read`-query entries in L. Hence, the history L is increased by the list of $2\log\ell - 1$ value $(k, \mathtt{R}, r_1)||...||(k, \mathtt{R}, r_{2\ell-1})$, where the indices r_j are ordered in decreasing order according to their location in the simulator's emulated tree. This perfectly mimics the real world behavior.

On input $(\mathtt{write}, i, x)$ **at interface** $\mathtt{C}_k$**:** Each write request can be divided into two two phases: (1) the retrieval of sub-tree $T_{\ell+i-1}$ and its validity check and (2) writing the updated sub-tree $T_{\ell+i-1}$ back to the server memory (except for the root N_0) in case the validity check was passed. Phase (1) is simulated as before by $\mathsf{sim}_{\text{auth}}$ and thus we focus on the second phase.

In the real system $\mathsf{auth}_{\mathcal{P}}[\mathbf{L}, \mathbf{SMR}^{k}_{\Sigma_1,2\ell}]$, the tree $T_{\ell+i-1}$ is updated locally and each node is subsequently written back to **SMR**. The

history HIST of the resource is thus increased by the following list of $2\log\ell - 1$ values, namely $(k, \mathtt{W}, r_1, N_{r_1})||...||(k, \mathtt{W}, r_{2\ell-1}, N_{2\ell-1})$.

In the ideal system, the procedure UPDATELOG will replace this entry in the history by the appropriate sequence of write-requests only if the write-request was successful. Note that the simulator $\mathsf{sim}_{\mathrm{auth}}$ is informed whether a client write resulted in a successful update (in which case $\mathrm{HIST}[q] = (k, \mathtt{W}, i, x)$) or whether the update failed (in which case $\mathrm{HIST}[q] = (k, \mathtt{Fail}, i, x)$).

On input `getHist` at interface S_H: In the real system, the output is the entire history of **SMR**. By the above analysis, a straightforward inductive argument shows that in system $\mathsf{sim}^{\mathsf{S}}_{\mathrm{auth}}\mathbf{aSMR}^{k}_{\Sigma,\ell}$, the simulator's simulated history L, which is output upon this query, emulates the real-world view perfectly.

On input $(\mathtt{write}, r, x)$ at interface S_I: An adversarial write request in the real world is a simple replacement of the memory cell r of **SMR**. If the value x corresponds to the last honest value written to this cell, then this operation might provoke that now certain sub-trees $T_{\ell+i-1}$ become valid again (and hence be involved in successful read and write requests).

This is simulated in the ideal world in that simulator $\mathsf{sim}_{\mathrm{auth}}$ checks, for $i = 0$ to $\ell - 1$, whether any sub-tree $T_{\ell+i-1}$ in its simulated server memory became valid again and issues $(\mathtt{restore}, i)$ to **aSMR** in this case.

In the other case, if the value x is unequal to the value N_r being replaced, this might lead to a couple of corrupted (logical) memory cells since certain sub-trees become invalid in the real system $\mathsf{auth}_{\mathcal{P}}[\mathbf{L}, \mathbf{SMR}^{k}_{\Sigma_1,2\ell}]$.

In the ideal world $\mathsf{sim}^{\mathsf{S}}_{\mathrm{auth}}\mathbf{aSMR}^{k}_{\Sigma,\ell}$, the simulator $\mathsf{sim}_{\mathrm{auth}}$ first updates its internal storage up to the current point by invoking UPDATELOG to get the actual value of N_r. If $N_r \neq x$, $\mathsf{sim}_{\mathrm{auth}}$ issues a $(\mathtt{delete}, i)$-query to **aSMR** for each location i whose tree $T_{\ell+i-1}$ got invalid due to this update.

This update, however, only simulates the real world perfectly if the change $N_r \leftarrow x$ led to an invalid sub-tree for the case $N_r \neq x$. This is the case, if the authentication of x is invalid or if timestamps do not

satisfy the invariant. The bad event, denoted by BAD occurs if the adversary manages to write a value x, that has never been written to location r and which nevertheless results in a valid sub-tree. We different two cases:

1. If location r stores a leaf node, this implies that x has the format (v', t'_r, tag') for which $t'_r > t_r$ *or* $(v' \neq v$ *and* $t'_r \geq t_r)$ holds, since for $t'_r < t'_r$ a valid sub-tree gets invalid since the overall invariant cannot longer hold (since the root value N_0 would be too large.) Hence, tag' corresponds to a valid forgery for the message (r, v', t_r) of $f_{sk}(.)$.
2. If location r stores an internal node, this implies that x has the format (t'_r, tag) with $t'_r > t_r$ (as otherwise it would constitute a reduction of the sum of the timestamps which will then eventually be smaller than N_0). In this case, tag corresponds to a forgery for the message (r, t'_r) of $f_{sk}(.)$.

Hence, we conclude that the real and ideal system are identical until event BAD occurs.

On input $(\texttt{read}, r)$ **at interface** S_H**:** In the real system, this query returns the current value at location r of **SMR**. This is either the last value written by any client interface or the last value written by the adversary. In the ideal system, the simulator updates its internal simulation of the server memory on each activation and hence, returns either the last value written to r according to its simulated history L or the value that was written by an adversarial write.

On inputs `startWriteMode` **and** `stopWriteMode` **at interface** `W`**:** First, in the real system $\mathsf{auth}_{\mathcal{P}}[\mathbf{L}, \mathbf{SMR}^{k}_{\Sigma_1,2\ell}]$, the first input allows the adversary to access and modify the server storage until the input `stopWriteMode` is input. The same holds for the the ideal system $\mathsf{sim}^{\mathsf{S}}_{\mathrm{auth}}\mathbf{aSMR}^{k}_{\Sigma,\ell}$, since the simulator does not react on adversarial queries at interface S_I in case INTRUSION = false and is allowed to access interface S_I of resource **aSMR** if and only if INTRUSION is set.

This concludes the analysis of the behavior. We see that the real system system and the ideal system are identical until event BAD occurs. In

particular, the occurrence of BAD implies a successful forgery against the MAC function $f_{sk}(.)$. We now construct and adversary $\mathcal{A} := \rho(\mathbf{D})$ from a distinguisher $\mathbf{D}$. The reduction emulates the real system towards the distinguisher $\mathbf{D}$ and uses the oracle provided by $\mathsf{EU\text{-}CMA}_f^{\mathrm{mac}}$ to evaluate the MAC-function. If $\mathbf{D}$ issues a write-query at interface S_I that provokes event BAD, $\rho(\mathbf{D})$ issues this value as a forgery to the game $\mathsf{EU\text{-}CMA}_f^{\mathrm{mac}}$. Hence, $\mathcal{A} := \rho(\mathbf{D})$ is an adversary for the MAC game and we observe that

$$\Delta^{\mathbf{D}}(\mathsf{auth}_{\mathcal{P}}[\mathbf{L}, \mathbf{SMR}^k_{\Sigma_1,2\ell}], \mathsf{sim}^{\mathsf{S}}_{\mathrm{auth}}\mathbf{aSMR}^k_{\Sigma,\ell}) \leq \Pr^{\mathbf{D}(\mathsf{auth}_{\mathcal{P}}[\mathbf{L},\mathbf{SMR}^k_{\Sigma_1,2\ell}])}[\mathsf{BAD}] \leq \mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{f,\mathcal{A}}.$$

This concludes the proof. □

6.4.2 Confidential from Authentic Server-Memory Resources

The protocol. We again specify two converters, which we call $\mathsf{init}_{\mathrm{priv}}$ (for initialization) and priv_{RW} (for the clients). Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a (CPA-secure) private-key encryption scheme with message space Σ, ciphertext space $\mathcal{C}$, and key space $\mathcal{K}$: To initialize, $\mathsf{init}_{\mathrm{priv}}$ executes Gen to get a key sk and stores the key in the local memory $\mathbf{L}$. To read and write to the authentic server-memory resource, the converters behave as follows: On input $(\mathtt{write}, i, x)$ at the outer interface, encrypt x and output $(\mathtt{write}, i, \mathsf{Enc}(sk, x))$ to $\mathbf{aSMR}$. If the write-operation returns ϵ (indicating an error), output ϵ at the outer interface. On input $(\mathtt{read}, i)$ at the outer interface, output $(\mathtt{read}, i)$ to $\mathbf{aSMR}$. If the received ciphertext is $c \neq \epsilon$, output $\mathsf{Dec}(sk, c)$ at the outer interface and ϵ otherwise. The protocol is described in detail in Figure 6.11.

Theorem 6.4.2. *Let $k, \ell \in \mathbb{N}$ and let Σ be an alphabet. The described protocol, i.e., the tuple of converters $\mathsf{priv} := (\mathsf{init}_{\mathrm{priv}}, \mathsf{priv}_{RW}, \ldots, \mathsf{priv}_{RW})$ (with a private-key encryption scheme $\mathcal{E}$ with ciphertext space $\mathcal{C}$) constructs the confidential (and authentic) server-memory resource $\mathbf{cSMR}^k_{\Sigma,\ell}$ from the authentic server-memory resource $\mathbf{aSMR}^k_{\mathcal{C},\ell}$ and a local private memory $\mathbf{L}$ (of constant size), with respect to the simulator $\mathsf{sim}_{\mathrm{priv}}$ as defined in Figure 6.12 and the pair $(\mathsf{honSrv}, \mathsf{honSrv})$. More specifically, we construct a reduction $\rho(\mathbf{D}) := \mathbf{DC}_I$ with a specific system $\mathbf{C}_I$ defined in the proof, such that for all distinguishers $\mathbf{D}$ and their associated*

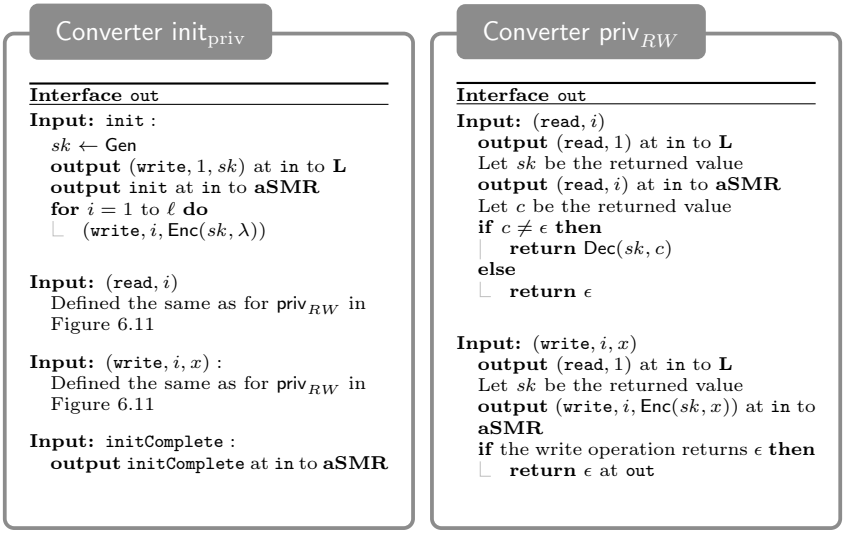


Figure 6.11: The initialization protocol (left) and the converter for the clients (right) to realize a confidential server memory from an authentic server memory.

CPA-adversaries $\mathcal{A} := \rho(\mathbf{D})$,

$$\Delta^{\mathbf{D}}(\mathsf{honSrv}^{\mathsf{S}}\ \mathsf{priv}_{\mathcal{P}}[\mathbf{L}, \mathbf{aSMR}^{k}_{\mathcal{C},\ell}], \mathsf{honSrv}^{\mathsf{S}}\mathbf{cSMR}^{k}_{\Sigma,\ell}) = 0$$

$$\textit{and} \qquad \Delta^{\mathbf{D}}(\mathsf{priv}_{\mathcal{P}}[\mathbf{L}, \mathbf{aSMR}^{k}_{\mathcal{C},\ell}], \mathsf{sim}^{\mathsf{S}}_{\text{priv}}\mathbf{cSMR}^{k}_{\Sigma,\ell}) = q \cdot \mathsf{Adv}^{\textit{ind-cpa}}_{\mathcal{E},\mathcal{A}},$$

where q is the total number of write operations at the client interfaces.

Proof Sketch. The correctness condition is again easy to verify. For the security condition, consider the simulator $\mathsf{sim}_{\text{priv}}$ in Figure 6.12 that generates an encryption key by its own and simulates the content for each write operation to be the encryption of the fixed value $\lambda \in \Sigma$. Furthermore, $\mathsf{sim}_{\text{priv}}$ simply forwards deletion-operations to **aSMR**. To argue about the security, a simple hybrid argument follows.

Let q be an upper bound on the number of write-queries at the client interfaces. For $i \in \{0, \ldots, q\}$, we define the system $\mathbf{H}_i$ that behaves as

Converter $\mathsf{sim}_{\mathrm{priv}}$

Initialization

$sk \twoheadleftarrow \mathsf{Gen}$
$L_{init} \leftarrow (0, \mathtt{W}, 1, \mathsf{Enc}(sk\lambda))||\ldots||(0, \mathtt{W}, \ell, \mathsf{Enc}(sk, \lambda))$
$pos \leftarrow 1$

Interface $\mathtt{S}_H$

Input: `getHist` :
  UpdateLog
  return L

Input: $(\mathtt{read}, i), i \in [\ell]$:
  UpdateLog
  if $S[i] = \epsilon$ **then**
    return $S[i]$
  else
    Determine the last entry in L that wrote a value c_i at location i
    return c_i

Interface $\mathtt{S}_I$ (INTRUSION = true)

Input: $(\mathtt{delete}, i) \in [\ell]$
  UpdateLog
  $S[i] \leftarrow \epsilon$
  output $(\mathtt{delete}, i)$ at `in` to **cSMR**

Input: $(\mathtt{restore}, i)$:
  $S[i] \leftarrow \perp$
  output $(\mathtt{restore}, i)$ at `in` to **cSMR**

procedure UpdateLog
  output `getHist` at `in` to **cSMR**
  Let Hist be the returned value
  for $j = pos$ to |Hist| **do**
    if Hist$[j] = (0, \mathtt{init})$ **then**
      $L \leftarrow L_{init}$
    else if Hist$[j] = (k, \mathtt{R}, i)$ **then**
      $L \leftarrow L||(k, \mathtt{R}, i)$
    else if Hist$[j] = (k, \mathtt{W}, i)$ **then**
      $L \leftarrow L||(k, \mathtt{W}, i, \mathsf{Enc}(sk, \lambda))$
    else if Hist$[j] = (k, \mathtt{Fail}, i)$ **then**
      $L \leftarrow L||(k, \mathtt{Fail}, i, \mathsf{Enc}(sk, \lambda))$
  $pos \leftarrow$ |Hist| $+ 1$

Figure 6.12: The simulator for the construction of a confidential memory.

$\mathsf{priv}_{\mathcal{P}}[\mathbf{L}, \mathbf{aSMR}^k_{\Sigma,\ell}]$ for the first i write-queries. However, for subsequent write-queries, not the real encrypted value is written to **aSMR**, but the encryption of λ. Hence, $\mathbf{H}_q$ is equivalent to $\mathsf{priv}_{\mathcal{P}}[\mathbf{L}, \mathbf{aSMR}^k_{\Sigma,\ell}]$ and H_0 is equivalent to $\mathsf{sim}^{\mathsf{S}}_{\mathrm{priv}}\mathbf{cSMR}^k_{\Sigma,\ell}$. Intuitively, since two adjacent hybrid systems $\mathbf{H}_{i-1}$ and $\mathbf{H}_i$ only differ in the way the ith write-query is encrypted (either the real value or λ), the overall security follows from the indistinguishability of ciphertexts.

To complete this last step, we define the reduction system $\mathbf{C}_i$ that behaves like $\mathbf{H}_i$, but instead of computing the encryptions and decryptions by itself, it queries the encryption and decryption oracles of game $\mathsf{IND\text{-}CPA}^b_{\mathcal{E}}$. In particular, on the jth write-query input $(\mathtt{write}, i, x)$, ask $\mathsf{IND\text{-}CPA}^b_{\mathcal{E}}$ for the encryption of x if $j < i$, ask $\mathsf{IND\text{-}CPA}^b_{\mathcal{E}}$ for the encryption of λ if $j > i$, and, in case $j = i$, challenge game $\mathsf{IND\text{-}CPA}^b_{\mathcal{E}}$ with input (x, λ) to receive the ciphertext. We immediately see that for all $i \in \{0, \ldots, q-1\}$,

$$\mathbf{H}_i = \mathbf{C}_i^{\mathsf{IND\text{-}CPA}^0_{\mathcal{E}}} = \mathbf{C}_{i+1}^{\mathsf{IND\text{-}CPA}^1_{\mathcal{E}}}. \tag{6.1}$$

Let $\mathbf{C}_I$ be the system that first chooses $i \in \{1, \ldots, q\}$ uniformly at random and then behaves as $\mathbf{C}_i$ and let us define $\mathcal{A} := \mathbf{DC}_I$. $\mathcal{A}$ is a valid adversary[4] for the CPA-Game and we observe that

$$\Pr\left[\mathcal{A}^{\mathsf{IND\text{-}CPA}^0_{\mathcal{E}}} = 1\right] = \frac{1}{q} \cdot \sum_{i=1}^{q} \Pr\left[\mathbf{DC}_i^{\mathsf{IND\text{-}CPA}^0_{\mathcal{E}}} = 1\right]$$

and

$$\begin{aligned}\Pr\left[\mathcal{A}^{\mathsf{IND\text{-}CPA}^1_{\mathcal{E}}} = 1\right] &= \frac{1}{q} \cdot \sum_{i=1}^{q} \Pr\left[\mathbf{DC}_i^{\mathsf{IND\text{-}CPA}^1_{\mathcal{E}}} = 1\right] \\ &= \frac{1}{q} \cdot \sum_{i=0}^{q-1} \Pr\left[\mathbf{DC}_i^{\mathsf{IND\text{-}CPA}^0_{\mathcal{E}}} = 1\right],\end{aligned}$$

where the last equality follows from equation (6.1).

[4]Note that the composition of the two systems $\mathbf{D}$ and $\mathbf{C}_I$, i.e., $\mathbf{DC}_I$, defines the behavior of the adversary and the way the oracles of the game are invoked.

Finally, we compute the distinguishing advantage by

$$\begin{aligned}
&\Delta^{\mathbf{D}}(\underbrace{\mathsf{priv}_{\mathcal{P}}[\mathbf{L}, \mathbf{aSMR}^{k}_{\Sigma,\ell}]}_{\mathbf{H}_q = \mathbf{C}_q^{\mathsf{IND\text{-}CPA}^0_{\mathcal{E}}}}, \underbrace{\mathsf{sim}^{\mathsf{S}}_{\mathrm{priv}}\mathbf{cSMR}^{k}_{\Sigma,\ell}}_{\mathbf{H}_0 = \mathbf{C}_0^{\mathsf{IND\text{-}CPA}^0_{\mathcal{E}}}}) \\
&= \Pr\left[\mathbf{DC}_q^{\mathsf{IND\text{-}CPA}^0_{\mathcal{E}}} = 1\right] - \Pr\left[\mathbf{DC}_0^{\mathsf{IND\text{-}CPA}^0_{\mathcal{E}}} = 1\right] \\
&= \sum_{i=1}^{q} \Pr\left[\mathbf{DC}_i^{\mathsf{IND\text{-}CPA}^0_{\mathcal{E}}} = 1\right] - \sum_{i=0}^{q-1} \Pr\left[\mathbf{DC}_i^{\mathsf{IND\text{-}CPA}^0_{\mathcal{E}}} = 1\right] \\
&= q \cdot \left(\Pr\left[\mathcal{A}^{\mathsf{IND\text{-}CPA}^0_{\mathcal{E}}} = 1\right] - \Pr\left[\mathcal{A}^{\mathsf{IND\text{-}CPA}^1_{\mathcal{E}}} = 1\right]\right) = q \cdot \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E},\mathcal{A}}.
\end{aligned}$$

This concludes the proof. □

6.4.3 Secure from Confidential Server-Memory Resources

We present an enhanced version of the Path ORAM protocol. The original Path ORAM protocol is due to Stefanov et al. [SDS+18]. In particular, we complement the original protocol with a proper error handling such that the protocol realizes the secure server-memory resource from an authentic and confidential server-memory resource.

Overview and notation. The protocol maintains a tree structure on the server-memory resource. For a logical memory with ℓ positions (assume ℓ is a power of two), the binary tree has height $L = \log(\ell)$ (and thus ℓ leaves). Each node N_r of the tree can hold Z memory blocks (where Z is a small constant greater or equal to 4 [SDS+18]). As usual, the tree is stored in the server memory in linear ordering from 1 to $2\ell - 1$, where in location 1 the root node N_1 is stored and where the leaves are located at addresses ℓ to $2\ell - 1$. We refer to the leaf node at address $\ell + i - 1$ as the ith leaf node. For such a leaf node, the unique path to the root of the tree is denoted $\mathcal{P}(i)$ and by $\mathcal{P}(i, lv)$ we denote the node at level lv on this path. The total number of blocks stored on the server is thus $Z \cdot (2\ell - 1)$.

The client stores a position map $\mathsf{position}$, which is a table of size $L \cdot \ell$ bits and maps all logical addresses to the index of its associated leaf node. At any time during protocol execution, the invariant holds that for any logical address $i \in [\ell]$, if $\mathsf{position}[i] = x$, then the correct data block (i, v)

Converter $\mathsf{init}_{\mathrm{sec}}$

Interface `out`

Input: `init` :
  Prepare $2\ell - 1$ nodes $N_r \leftarrow (0, \lambda)^Z$
  $S \leftarrow \emptyset$
  position $\leftarrow$ empty list with a capacity of ℓ items
  for $i = 1$ to ℓ **do**
    $S \leftarrow S \cup \{(i, \lambda)\}$
    position$[i] \twoheadleftarrow \{1, \ldots \ell\}$
  for $x = 1$ to ℓ **do**
    for $lv = L, L-1, \ldots, 0$ **do**
      Let r be the memory address of node $\mathcal{P}(x, lv)$
      Let d be the number of dummy blocks in node N_r
      $S' \leftarrow \{(i', v') \in S \mid \mathcal{P}(x, lv) = \mathcal{P}(\mathsf{position}[i'], lv)\}$
      $m \leftarrow \min\{|S'|, Z, d\}$
      Truncate S' to m blocks
      $S \leftarrow S \setminus S'$
      Denote the blocks in S' by b_i, $i = 1 \ldots m$
      for $k = 1$ to Z **do**
        if $N_r[k] = (0, \lambda)$ **then**
          $N_r[k] \leftarrow b_k$
  for all nodes N_r in decreasing order of index **do**
    output $(\mathtt{write}, r, N_r)$ at `in` to **cSMR**

Input: $(\mathtt{read}, i)$ or $(\mathtt{write}, i, x)$:
  Defined the same as for $\mathsf{sec}_{\mathrm{RW}}$ in Figure 6.14

Input: `initComplete` :
  output `initComplete` at `in` to **cSMR**

Figure 6.13: The initialization protocol for the construction of a secure server memory.

is contained in a node on the path $\mathcal{P}(x)$ or in the stash S. The stash is a local buffer maintained by the client that stores data blocks that overflow during the protocol execution. A data block overflows if all suitable nodes in the tree are already occupied by real memory blocks. The number of overflowing blocks is proven to be small in [SDS+18].

Protocol. Initially, the tree is initialized to contain ℓ empty blocks of the form (i, λ) for each address $i \in [\ell]$. Upon initialization, the tree is built to contain these empty blocks. In addition, the position table and the stash are stored in the shared memory $\mathbf{L}$ and to each address i, a

uniformly random leaf node is assigned, i.e., $\mathsf{position}[i] \twoheadleftarrow \{1, \dots \ell\}$. Since each node of the tree should be a list of exactly Z elements, each node is complemented with the necessary amount of dummy elements which we encode as $(0, \lambda)$ (as opposed to real elements that contain the normal addresses and the associated data block). The entire tree is then written to the server storage. We give the formal description of converter $\mathsf{init}_{\mathrm{sec}}$ in Figure 6.13. To access a logical address i to either read or update the corresponding value v, the client reads the associated index of the leaf node $x \leftarrow \mathsf{position}[i]$ and reassigns $\mathsf{position}[i]$ to a new uniformly random leaf. Next, the client retrieves all nodes on the path $\mathcal{P}(x)$ from the server memory (from leaf to root) and all found real elements (j, v) $(j > 0)$ are added to the stash. In case the value at position i is to be updated, it is assigned a new value at this point. Finally, the nodes of $\mathcal{P}(x)$ are newly built and written back to the server. In this write-back phase, as many blocks as possible from the local stash are "pushed" onto this path. To deal with failures on a read or write-access to a logical address i, the protocol behaves as follows: if during the above execution, a read request to the server is answered by ϵ, indicating that a node is deleted, then the logical address i is marked as invalid in the local position table $\mathsf{position}[i] \leftarrow \epsilon$. To remain oblivious in this case, the protocol subsequently writes back all previously retrieved nodes without any modifications (yielding a sequence of dummy accesses). In a subsequent request to retrieve logical block i, the protocol will detect the invalid entry in the position table and just return ϵ. To remain oblivious, the protocol additionally reads a uniformly random path from the outsourced binary tree and subsequently re-writes the very same elements without modifications (again yielding a sequence of dummy accesses). If during these dummy accesses an error occurs, i.e., the server-memory resource returns ϵ upon a request, this is simply ignored. This concludes the description of the protocol. A more precise specification can be found Figure 6.14. We denote this client converter by $\mathsf{sec}_{\mathrm{RW}}$. The security of the protocol is assured by the following theorem. It implies that the above error-handling for Path ORAM is sufficient to realize the secure server-memory resource and to ensure strong security guarantees.

Theorem 6.4.3. *Let $k, \ell, Z \in \mathbb{N}$ and $\Sigma_1 := ((\{0\} \cup [\ell]) \times \Sigma)^Z$ for some finite non-empty set Σ. The above described protocol $\mathsf{sec} := (\mathsf{init}_{\mathrm{sec}}, \mathsf{sec}_{\mathrm{RW}}, \dots, \mathsf{sec}_{\mathrm{RW}})$ (with k copies of $\mathsf{sec}_{\mathrm{RW}}$) constructs the secure*

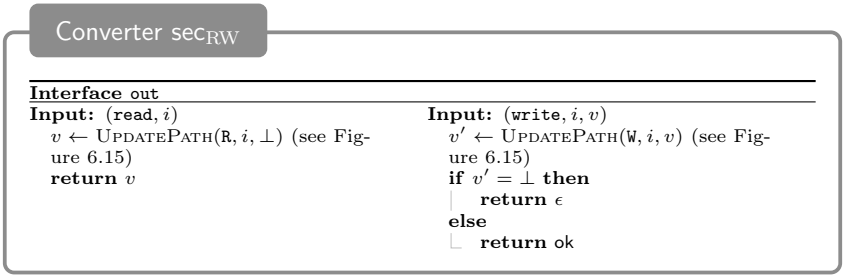


Figure 6.14: The converter for the clients to realize a secure server memory from a confidential and authentic server memory.

server-memory resource $\mathbf{sSMR}^{k,1}_{\Sigma,\ell}$ *from the confidential (and authentic) server-memory resource* $\mathbf{cSMR}^{k}_{\Sigma_1,2\ell}$ *and a local memory, with respect to the simulator* $\mathsf{sim}_{\mathrm{sec}}$ *described in Figure 6.16 and the pair* $(\mathsf{honSrv}, \mathsf{honSrv})$. *More specifically, for all distinguishers* $\mathbf{D}$

$$\Delta^{\mathbf{D}}(\mathsf{honSrv}^{\mathsf{S}}\ \mathsf{sec}_{\mathcal{P}}[\mathbf{L}, \mathbf{cSMR}^{k}_{\Sigma_1,2\ell}], \mathsf{honSrv}^{\mathsf{S}}\mathbf{sSMR}^{k,1}_{\Sigma,\ell}) = 0$$
$$\textit{and} \qquad \Delta^{\mathbf{D}}(\mathsf{sec}_{\mathcal{P}}[\mathbf{L}, \mathbf{cSMR}^{k}_{\Sigma_1,2\ell}], \mathsf{sim}^{\mathsf{S}}_{\mathrm{sec}}\mathbf{sSMR}^{k,1}_{\Sigma,\ell}) = 0.$$

Proof. We prove the security condition and again analyze the input-output behavior of both systems involved. To this end, we consider the possible inputs at each interface.

On input `init`, `initComplete` at interface C_0: On input `init` to the real system, i.e., to $\mathsf{sec}_{\mathcal{P}}[\mathbf{L}, \mathbf{cSMR}^{k}_{\Sigma_1,2\ell}]$, the converter $\mathsf{init}_{\mathrm{sec}}$ first initializes the position map position by assigning to each logical address i the corresponding leaf number uniformly at random. Then, the converter stores all initial blocks (i, λ) for $i = 1 \ldots \ell$ in the stash S. Subsequently, the binary tree T consisting of nodes N_0 to $N_{2\ell-1}$ (in the usual linear ordering) is locally built: each path from any leaf to the root is examined and as many blocks as possible are pushed from the stash S to a node in the tree. After this step, a block (i, λ) is either found in the stash S (stored in the local memory) or within the tree in any of the nodes of the path $\mathcal{P}(i)$. Finally, the whole tree is written to the server-memory resource (in decreasing order of the

Function UpdatePath

```
1: function UpdatePath(op, i, v')
2:     Res ← ⊥
3:     Retrieve the stash S and the position table position from L
4:     x ← position[i]
5:     if x ≠ ϵ then
6:         position[i] ↞ {1, ..., ℓ}
7:         for lv = L, L − 1, ..., 0 do
8:             Let r be the memory address of node P(x, lv)
9:             output (read, r) at in to cSMR
10:            Store the returned value as N_r
11:        if all fetched nodes N_r ≠ ϵ and P(x) is not marked as invalid then
12:            for each node N_r do
13:                Parse N_r as a list of Z blocks b_i ∈ {(i, v) | i ∈ ℕ} ∪ {λ}
14:                for i = 1 to Z do
15:                    if b_i ≠ (0, λ) then
16:                        S ← S ∪ {b_i}
17:            Retrieve block b from S such that b = (i, v) for some v ∈ Σ
18:            Res ← v
19:            if op = W then
20:                Replace (i, v) in S by (i, v')
21:            for lv = L, L − 1, ..., 0 do
22:                Let r be the memory address of node P(x, lv)
23:                N ← [ ]
24:                S' ← {(i', v') ∈ S | P(x, lv) = P(position[i'], lv)}
25:                m ← min{|S'|, Z}
26:                Truncate S' to m blocks
27:                S ← S \ S'
28:                Denote the blocks in S' by b_i, i = 1 ... m
29:                for k = 1 to Z do
30:                    if k ≤ m then
31:                        N ← N||b_k
32:                    else
33:                        N ← N||(0, λ)
34:                output (write, N, r) at in to cSMR
35:                if the write query returns ϵ then
36:                    Mark all paths containing P(x, lv) as invalid in position table.
37:        else
38:            position[i] ← ϵ
39:            for lv = L, L − 1, ..., 0 do
40:                Let r be the memory address of node P(x, lv)
41:                output (write, r, N_r) at in to cSMR
42:    else                ▷ Simulate dummy accesses if logical address i is marked invalid.
43:        x ↞ {1, ..., ℓ}
44:        for lv = L, L − 1, ..., 0 do
45:            Let r be the memory address of node P(x, lv)
46:            output (read, r) at in to cSMR
47:            Store the returned value as N_r
48:        for lv = L, L − 1, ..., 0 do
49:            Let r be the memory address of node P(x, lv)
50:            output (write, r, N_r) at in to cSMR
51:    Store the stash S and the position table position in L
52:    return Res
```

Figure 6.15: Definition of the path update function.

Converter $\mathsf{sim}_{\mathrm{sec}}$

Initialization

for $i = 1$ to $2\ell - 1$ **do**
  $N_i \leftarrow \texttt{valid}$
Let T be the binary tree consisting of nodes $N_0, \ldots, N_{2\ell-1}$ in linear ordering
$L_{init} \leftarrow (0, \texttt{init})||(0, \texttt{W}, 2\ell - 1)|| \ldots ||(0, \texttt{W}, 1)$
$pos \leftarrow 1$
$L \leftarrow [\,]$

Interface $\mathtt{S}_H$

Input: $\texttt{getHist}$:
  UpdateLog
  return L

Input: $(\texttt{read}, r), r \in [2\ell - 1]$:
  UpdateLog
  return λ

Interface $\mathtt{S}_I$ (INTRUSION = true)

Input: $(\texttt{delete}, r) \in [2\ell - 1]$
  UpdateLog
  $N_r \leftarrow \texttt{invalid}$
  $I \leftarrow \{i \in [\ell] \mid \text{Path } \mathcal{P}_T(i) \text{ contains at least one invalid node}\}$
  $\alpha \leftarrow \frac{|I|}{\ell}$
  output $(\texttt{pollute}, \alpha)$ at $\texttt{in}$ to **sSMR**

Input: $(\texttt{restore}, r)$:
  UpdateLog
  $I_{\mathrm{old}} \leftarrow \{i \in [\ell] \mid \text{Path } \mathcal{P}_T(i) \text{ contains at least one invalid node}\}$
  $N_i \leftarrow \texttt{valid}$
  $I_{\mathrm{new}} \leftarrow \{i \in [\ell] \mid \text{Path } \mathcal{P}_T(i) \text{ contains at least one invalid node}\}$
  $\delta \leftarrow \frac{|I_{\mathrm{old}}| - |I_{\mathrm{new}}|}{\ell}$ ▷ $I_{\mathrm{new}} \subseteq I_{\mathrm{old}}$
  output $(\texttt{reducePollution}, \delta)$ at $\texttt{in}$ to **sSMR**

---

procedure UpdateLog
  output $\texttt{getHist}$ at $\texttt{in}$ to **sSMR**
  Let Hist be the returned value
  for $j = pos$ to $|\text{Hist}|$ **do**
    if $\text{Hist}[j] = (0, \texttt{init})$ **then**
      $L \leftarrow L_{init}$
      $I \leftarrow \emptyset$
    else if $\text{Hist}[j] = (k, \texttt{Access})$ **then**
      $I \leftarrow \{i \in [\ell] \mid \text{Path } \mathcal{P}_T(i) \text{ contains only valid nodes}\}$
    else if $\text{Hist}[j] = (k, \texttt{Failed})$ **then**
      $I \leftarrow \{i \in [\ell] \mid \text{Path } \mathcal{P}_T(i) \text{ contains at least one invalid node}\}$
    if $I \neq \emptyset$ **then**
      $x \twoheadleftarrow I$
      for $lv = L, L-1, \ldots, 0$ **do** ▷ Simulate read access.
        Let r be the address of simulated node $N_r = \mathcal{P}_T(x, lv)$
        $L \leftarrow L||(k, \texttt{R}, r)$
      for $lv = L, L-1, \ldots, 0$ **do** ▷ Simulate write access.
        Let r be the address of simulated node $N_r = \mathcal{P}_T(x, lv)$
        if $N_r = \texttt{valid}$ **then**
          $L \leftarrow L||(k, \texttt{W}, r)$
        else
          $L \leftarrow L||(k, \texttt{Fail}, r)$
  $pos \leftarrow |\text{Hist}| + 1$

Figure 6.16: The simulator for the construction of a secure memory.

linear index), which adds $2\ell - 1$ entries $(0, \mathtt{W}, r)$ for $r = 2\ell - 1 \ldots 1$ to the history.

In the ideal system $\mathsf{sim}^{\mathsf{S}}_{\mathrm{sec}}\mathbf{sSMR}^{k,1}_{\Sigma,\ell}$, the command sets the value of any storage location to λ and adds the initial entry $(0, \mathtt{init})$ to the history. The simulator will replace this first entry by the list L_{init} that contains the simulated accesses that would happen in the real world. This perfectly emulates the real-world view.

Finally, on input `initComplete`, both systems deactivate interface $\mathtt{C}_0$ and the other client interfaces are operational from this point onwards.

On input $(\mathtt{read}, i)$ **at interface** $\mathtt{C}_k$**:** Upon this input at a client interface of the real system $\mathsf{sec}_{\mathcal{P}}[\mathbf{L}, \mathbf{cSMR}^{k}_{\Sigma_1,2\ell}]$, the protocol executes a write access to the memory resource. Inspecting the program code in Figure 6.14, the function UPDATEPATH is executed in read-mode, i.e., where the operation $op = \mathtt{R}$ (and hence no other arguments need to be specified). First, the current leaf node of address i is read from the position table (line 2). The program now branches into two tracks: if a valid leaf index x is returned, then the instructions on lines 5 to 41 are executed. The other case corresponds to the event that a previous access to logical address i was invalid and lines 42 to 50 are executed instead. Let us focus on the successful branch first: the client downloads all nodes corresponding to the path from the xth leaf node to the root. The accessed path is determined in a uniformly random way, since each time a path for logical address i is successfully accessed, a new uniformly random value is written to the position and determines to be accessed the next time when address i is to be read (line 6). If all retrieved nodes are valid, i.e., if the test on line 11 is passed, all the blocks contained in the nodes are added to the local stash S (lines 12 to 16) and finally the retrieved value is read from the stash (lines 17 and 18). To conclude this operation, the updated path is written back to the confidential server memory (lines 21 to 34). The update step tries to push as many blocks as possible from the stash into the tree nodes. Only the blocks (j, v) can be inserted into a node in the intersection of $\mathcal{P}(x)$ and $\mathcal{P}(j)$ (condition on line 24). However, if the test on line 11 is not passed, i.e., if an invalid node is retrieved, then the currently read logical block is declared as invalid by setting the position table

$\mathsf{position}[i] \leftarrow \epsilon$ on line 38. Furthermore, the client simply writes back the nodes it just retrieved without modification. Some of these writes might not be successful but this can safely be ignored (as nothing is changed). Overall, we conclude that this branch adds in any case $\log(\ell)$ read-requests and $\log(\ell)$ write-requests to the history of **cSMR**.

The second branch is taken if the position i is known to have failed in the past (lines 42 to 50). Then, the protocol simply sends $\log(\ell)$ read-requests to the server to retrieve a randomly chosen path and then rewrites the path unaltered. This adds another $\log(\ell)$ write-request to the history.

Overall, the probability that an access to logical address i is successful given there has not been an invalid access[5] since initialization, is exactly the ratio of the number of valid paths and all ℓ paths. Similarly, the probability that an access to logical address i is invalid given there has not been an invalid access since initialization, is exactly the ratio of the number of invalid paths and all ℓ paths. In any other case, an access to logical address i will return ϵ with probability one. Finally, we observe that on input $(\mathtt{read}, i)$ each of the ℓ paths of the tree has equal probability to be accessed.

Let us now consider the ideal system $\mathsf{sim}^{\mathsf{S}}_{\mathrm{sec}}\mathbf{sSMR}^{k,1}_{\Sigma,\ell}$. Upon a read-query, we again have two possible branches. This is seen by inspecting the program code of **sSMR** on a client-read request at interface C_k for $k > 0$.[6] Given that there has never been an invalid access to address i, the probability of a successful access is exactly $1 - \alpha$, and that of an invalid access is exactly α, where α is the pollution factor that can be set by the simulator. In each step of the execution, the simulator $\mathsf{sim}_{\mathrm{sec}}$ maintains the invariant that α equals the ratio of invalid paths and all ℓ possible paths. In particular, as explained below, on each deletion-query by the distinguisher, the simulator updates the parameter α accordingly.

We now look at how the simulator simulates the real-world memory access and maintains the simulated history. The simulator is informed, whether an operation was evaluated to be successful (entry

[5] We mean an access that returned ϵ.

[6] Recall that for the sake of simplicity (and without loss of generality), we do not assume any failure during the initialization phase.

$(k, \texttt{Access})$ in the history), or whether it was evaluated to be a fail (entry $(k, \texttt{Failed})$ in the history). In the first case, the simulator chooses a random path from all the paths that only contain valid nodes. (The statistics which nodes are valid and which are not is maintained as explained below for input $(\texttt{delete}, r)$ to interface $\texttt{S}_I$). In the second case, the simulator simulates the accesses to a random path from the set of all paths that contain at least one invalid node. Overall, this means that on input $(\texttt{read}, i)$ to resource **sSMR**, the probability for any fixed path to be added to the history is $\frac{1}{\ell}$. This is easily seen by a case distinction: the probability that a particular valid path is added to the history is $(1-\alpha) \cdot \frac{\#\text{valid paths}}{\ell}$. For $\alpha = \frac{\#\text{invalid paths}}{\ell}$ this gives us a probability of $\frac{1}{\ell}$ for all valid paths. The other case is analogous and we see that on each read-request, a uniformly random path is added to the history.

We can conclude that the behavior of the simulator mimics the real world behavior. In particular, the simulated history is updated accordingly such that the failure probabilities are identical, as well as the distribution of the access pattern in the simulated history.

On input $(\texttt{write}, i, x)$ **at interface** $\texttt{C}_k$**:** On a write-instruction to the systems, the same function UPDATEPATH is executed, but with arguments $op = \texttt{W}$, i and v', where v' is the new value for address i. The code for this case is identical to the read case except for the instructions on lines 19 and 20. Since these two lines do not affect the observable behavior, the analysis of this case follows from the analysis of the previous analysis of the read-instructions.

On input `getHist` **at interface** $\texttt{S}_H$**:** In the real system, the output is the history of **cSMR**. By the above analysis, a straightforward inductive argument shows that in case of system $\mathsf{sim}^{\mathsf{S}}_{\text{sec}}\mathbf{sSMR}^{k,1}_{\Sigma,\ell}$, the simulator's internally maintained history L, which is output upon this query, emulates the real-world view perfectly.

On input $(\texttt{restore}, r)$ **at interface** $\texttt{S}_I$**:** In the real system, the restore operation makes a node, which was invalid before, become valid again. This means that the number of valid paths might increase. In fact, for all logical address i, that have not failed on any access so far, the probability thus increases that the next read or write

request is successful. The already failed addresses are not affected by this change since the local position table is not affected by a restore command.

In the ideal system, the simulator updates the pollution factor α of the server memory **sSMR** accordingly by recomputing the ratio of invalid paths after the node N_r becomes valid again (note that this ratio will not increase). Hence, in both worlds, the effects of a restore command are identical.

On input $(\texttt{delete}, r)$ **at interface** $\mathtt{S}_I$**:** In the real system, the delete operation makes a node, which was valid before, become invalid. This means that the number of invalid paths increases. In fact, for all logical address i, that have not failed on any access so far, the probability thus increases that the next read or write request fails. The already failed addresses are not affected by this change since the local position table is not affected by a deletion command.

In the ideal system, the simulator updates the pollution factor α of the server memory **sSMR** by recomputing the ratio of invalid paths after the node N_r becomes invalid again. Hence, in both worlds, the effects of a deletion command are identical.

On input $(\texttt{read}, r)$ **at interface** $\mathtt{S}_H$**:** On any command $(\texttt{read}, r)$ both systems simply return the dummy symbol λ. This holds by definition of system **cSMR** in the real world and by definition of the simulator $\mathsf{sim}_{\mathrm{sec}}$ in the ideal world.

On inputs `startWriteMode` **and** `stopWriteMode` **at interface** `W`**:** In case of the real system $\mathsf{sec}_{\mathcal{P}}[\mathbf{L}, \mathbf{cSMR}^{k}_{\Sigma_1,2\ell}]$, the first input allows the adversary to access and modify the server storage until the input `stopWriteMode` is input. The same holds for the the ideal system $\mathsf{sim}^{\mathsf{S}}_{\mathrm{sec}}\mathbf{sSMR}^{k,1}_{\Sigma,\ell}$, since the simulator does not react on adversarial queries at interface $\mathtt{S}_I$ in case INTRUSION = false and is allowed to access interface $\mathtt{S}_I$ of resource **sSMR** if and only if INTRUSION is set.

This ends our analysis of the behavior. We conclude that on each input, the observable effects are identical for the real system and the ideal system. The statement follows. □

Client-side storage reduction. At first sight, the client storage overhead seems unpractical since the size of the position map is $\ell \log(\ell)$ bits, which corresponds roughly to ℓ data blocks if we assume that each data block has a size B of (at least) $\log(\ell)$ bits. There are a couple of techniques suggested to reduce this storage overhead. Stefanov et al. [SSS12] describe that under realistic workloads such as mostly sequential accesses in file systems, the position table might consume only 0.13ℓ bytes, achieved for example by just maintaining a counter per block instead of its random position and to infer the block's position via a PRF applied to the current counter and the block number. This suggests that even for an an outsourced storage in the order of a couple of terabytes, the position table would not exceed one gigabyte. A second technique to reduce the client storage overhead is by outsourcing the position itself in a clever way. However, not all schemes are equally suitable as will be discussed in the next section.

Improving the resilience by replication. There is a simple protocol that improves the resilience to losing data blocks. The protocol stores each data block t times within the secure server memory. Formally, this protocol constructs resource $\mathbf{sSMR}^{k,t}_{\Sigma,\ell}$ from $\mathbf{sSMR}^{k,1}_{\Sigma,t\cdot\ell}$. Recall that in the former resource, only failing to read (or write) a logical memory cell more than t times implies that the data block is not accessible any more. We sketch the converter for initialization, denoted $\mathsf{init}_{\mathrm{rep},t}$ and the client converter rep_t.

On input `init` at the outer interface of $\mathsf{init}_{\mathrm{rep},t}$, output `init` to $\mathbf{sSMR}^{k,1}_{\Sigma,t\cdot\ell}$ and additionally store for each $i \in \ell$ the value c_i (initially zero) in the local storage $\mathbf{L}$. The value c_i denotes the number of failed accesses to logical address i. On input $(\texttt{read}, i)$ to converter $\mathsf{init}_{\mathrm{rep},t}$ or rep_t, output $(\texttt{read}, i + \min\{c_i, t-1\})$ and return whatever is returned by resource $\mathbf{sSMR}^{k,1}_{\Sigma,t\cdot\ell}$. In case ϵ is returned, the converter sets $c_i \leftarrow c_i + 1$. On input $(\texttt{write}, i, x)$ to converter $\mathsf{init}_{\mathrm{rep},t}$ or rep_t output $(\texttt{write}, i + r, x)$ to $\mathbf{sSMR}^{k,1}_{\Sigma,t\cdot\ell}$ for all $r = 0 \ldots t-1$ and output ϵ at the outer interface for each failed write access to the resource (and ok for the others). For this protocol, one can show the following lemma:

Lemma 6.4.4. *Let $k, \ell, t \in \mathbb{N}$. be a secure server-memory resource with the usual parameters. The above described replication protocol* $\mathsf{rep} := (\mathsf{init}_{rep,t}, \mathsf{rep}_t, \ldots, \mathsf{rep}_t)$ *(with k copies of rep_t) constructs the*

secure server-memory resource $\mathbf{sSMR}^{k,t}_{\Sigma,\ell}$ *from the secure server-memory resource* $\mathbf{sSMR}^{k,1}_{\Sigma,t\cdot\ell}$. *More specifically, there is a simulator* $\mathsf{sim}_{\mathrm{rep}}$ *such that for all distinguishers* $\mathbf{D}$*,*

$$\Delta^{\mathbf{D}}(\mathsf{honSrv}^{\mathsf{S}}\ \mathsf{rep}_{\mathcal{P}}[\mathbf{L}, \mathbf{sSMR}^{k,1}_{\Sigma,t\cdot\ell}], \mathsf{honSrv}^{\mathsf{S}}\mathbf{sSMR}^{k,t}_{\Sigma,\ell}) = 0$$
$$\textit{and} \qquad \Delta^{\mathbf{D}}(\mathsf{rep}_{\mathcal{P}}[\mathbf{L}, \mathbf{sSMR}^{k,1}_{\Sigma,t\cdot\ell}], \mathsf{sim}^{\mathsf{S}}_{\mathrm{rep}}\mathbf{sSMR}^{k,t}_{\Sigma,\ell}) = 0.$$

6.4.4 Do all ORAM Schemes realize a Secure Server-Memory Resource?

Our formalization provides strong security guarantees. Especially, the failure probabilities are required to be independent and the same for each memory location. However, not all existing ORAM schemes satisfy this level of security. We elaborate on two popular ORAM schemes. We show that in the recursive Path ORAM scheme by Stefanov et al. [SSS12], failures among memory locations are correlated. In the case of the Goodrich-Mitzenmacher ORAM scheme [GM11], we show that the failure probabilities are not the same for all (logical) memory locations. As we explain, the latter property is structural and therefore applies also to stronger notions, for example ORAM schemes that satisfy the NRPH-property of [CKW13].

The recursive Path ORAM scheme. A beautiful technique to reduce the client storage overhead is by using the Path ORAM scheme recursively as suggested by Stefanov et al. [SSS12]. In recursive Path ORAM, the position table itself is outsourced using another (and smaller) instance of a Path ORAM scheme. This smaller instance could itself outsource its position table to an even smaller ORAM scheme etc. The final instance (i.e., the base case), stores its position table in the local memory. Assuming a constant block size $B > \log(n)$, each recursive instance reduces the number of positions by a factor $f := \frac{B}{\log(n)} > 1$, where each block is used to store (roughly) f entries of the position table. Hence, after recursion depth in the order of $O(\log(\ell))$, the position table stored in the client storage is of size roughly $\log(\ell)$ data blocks. A formal proof of this is given in [SSS12].

Let us first describe one (recursion) step of this procedure in our formalism: We consider the similar scenario as before, but we replace

the local memory $\mathbf{L}$ by an instance of a secure server-memory resource $\mathbf{sSMR}$. This additional secure server storage memory has $\ell' < \ell$ storage locations, each of which holds a tuple of f values of the position table $\mathsf{position}$. The protocols need to be adapted only slightly: let the converter $\mathsf{init}'_{\mathrm{sec}}$ be defined as $\mathsf{init}_{\mathrm{sec}}$ except that the position table is written to secure server-memory resource instead of the private memory $\mathbf{L}$. Let further $\mathsf{sec}'_{\mathrm{RW}}$ be defined as converter $\mathsf{sec}_{\mathrm{RW}}$ but instead of the instruction $x \leftarrow \mathsf{position}[i]$, the converter computes $q \leftarrow (i-1) \operatorname{div} f$ and sends a read instruction $(\mathtt{read}, q+1)$ to the secure memory to obtain the tuple $(\mathsf{position}[fq+1], \ldots, \mathsf{position}[f(q+1)-1])$, where the desired value x is at position $i - qf$ in the tuple. Similarly, the subsequent update step $\mathsf{position}[i] \leftarrow x$ now consists of first updating the tuple at the respective location and then sending a write instruction to the secure memory resource to write the entire tuple back to location $q+1$.

The question now is: does the protocol still realize a secure server-memory resource? Unfortunately, the answer to this question is negative. On an intuitive level, the reason is that logical memory addresses are grouped in blocks. For example, the logical memory locations $i = 1 \ldots f$, i.e., their mappings $\mathsf{position}[1] \ldots \mathsf{position}[f]$, are an atomic block in the recursive Path ORAM scheme. This, however, implies that if the lookup fails for one logical address in $\mathsf{sec}'_{\mathrm{RW}}$, then it fails for all the others in that block as well. For the overall scheme, this means that failing to access the value at location $i = 1$ is not independent of failing to access the value at location $i = 2$ etc. In contrast, failing to access the value at location $f+1$ is again independent, as it resides in a different block of the smaller ORAM scheme. It is easy to exploit this observation to design a distinguisher that distinguishes the system[7] $\mathsf{sec}'_{\mathcal{P}}[\mathbf{sSMR}, \mathbf{cSMR}]$ and its ideal goal, the desired secure memory resource, with noticeable advantage. This scheme thus only constructs a weaker variant of the resource, where failures among data blocks are correlated.

ORAM schemes based on a hierarchical structure of hash-tables. A prominent ORAM scheme in this category is due to Goodrich and Mitzenmacher [GM11] which is based on cuckoo-hashing and follows the hierarchical approach envisioned by Goldreich and Ostrovsky [GO96]. The hierarchical approach organizes the data in levels, where each level is

[7] We omit here the parameters of the systems for brevity.

capable of storing two times as many elements as the level above. The first level can be thought of as an array of small size. The lowest level is capable of storing ℓ elements (and hence the number of levels is in $O(\log(\ell))$. Each level except the first is either a standard hash-table or a cuckoo hash-table[8] An element is encoded in the familiar form (i, v), where v denotes the value at logical address i. On a given level lv, if the pair resides in the table of that level, then it is found at location $H^{lv}(i)$, where H is a hash function.[9] To perform a search for an address i, each level lv is accessed (starting from the top level) and the location $H^{lv}(i)$ is read until a pair (i, v) is found. After the element has been found, the remaining tables are accessed at uniformly random locations. After all accesses have been performed, the pair is inserted into the top level array. We denote this random walk through the tables succinctly by $RW(i)$ and understand the above procedure. Inserting the element into the top level is a crucial step: intuitively, the access pattern does not reveal any information, since after each successful search (to random locations from the servers point of view), the element will be found in the top level in a subsequent search and the accesses to lower tables still look random. Overall, no lookup for an address i (accessing position $H^{lv}(i)$ on level lv) is performed twice for the same table. To maintain this invariant, and to prevent tables from overflowing, periodic rebuild phases occur. Such a rebuild phase servers two purposes: first, it moves data from one level to the next lower level in an oblivious way. This has the effect that infrequently accessed items are more likely to be found in lower tables. Second, new hash functions $H^{lv}(.)$ are chosen for the levels to keep the access pattern random looking. Both of these steps (regular re-hashing and moving elements downwards in the hierarchy) are crucial to prove its security in a passive setting [KLO12, GM11] or in an active setting where the protocol aborts upon detecting an error.

Similar to the construction presented in Section 6.4.3, it seems that each access is essentially a random walk from the top-level to the bottom level and one would probably expect that we have equal failure probability on any search for the item with logical address i. More precisely, we say that an access to address i is successful if it has never failed in the past and the current random walk $RW(i)$ does not access any deleted cell. Thus,

[8] Usually the smaller levels are implemented using standard hash tables and larger levels are implemented using cuckoo hash-tables.

[9] H is usually modeled as a uniform random function.

one could suspect that a similar adaption of the GM scheme would realize **sSMR** from a confidential and authentic memory. However, this is not the case as one can see using the following thought experiment which lets us conclude that there is a strategy that makes elements of tables higher in the hierarchy fail with significantly smaller probability than elements residing towards the bottom of the hierarchy. Imagine a hierarchy of tables and assume that the address-value pair $(1, v)$ is currently stored at some level lv and that no failure has occurred so far. We assume v to be a uniform random value of some alphabet. Consider an attacker that now deletes a uniform random location of the hash-table at level lv: with noticeable probability, this will hit exactly the pair $(1, v)$ and thus delete the information which value is associated to logical address 1. If this event happens, no search will ever be able to output v (except with negligible probability). Now, imagine that (roughly) 2^{lv} read operations for a different logical address, say 2, are performed. This number of accesses is sufficient to provoke a rebuild phase that moves all elements contained in level lv to the table at the lower lever. However, if the pair $(1, v)$ was deleted before, the lower level tables cannot contain the correct value for address 1 and hence any subsequent access to this address cannot return a consistent value. We can thus conclude: the probability that at this point in time, accessing logical address 1 returns the correct (and valid) value v is equal to the sum of the two probabilities that the actual random walk through the tables is valid *and* the probability that it has not been deleted before this last rebuild phase.

$$\begin{aligned} &\Pr[\text{Access to address 1 returns correct result}] \\ &= \Pr[\text{RW}(1) \text{ is valid}] + \Pr[\text{Pair } (1, v) \text{ was not deleted before at level lv}]. \end{aligned} \tag{6.2}$$

Hence, the probability is not uniform for all data blocks: in fact, items stored at levels lower than lv have a significant lower probability of failing than items stored at levels greater than lv, since they do not have this additional error term on the right hand side. For example, the probability of a read operation returning an invalid value is not the same for locations 1 and 2 for the above attacker strategy.

The above observation is likely to have an impact in practical settings, where the ORAM scheme is used as part of a larger application. Assume that the application stores some control information in the memory at a known locations (e.g., address 1) and does not frequently update this

location such that the above considerations apply. Then, an attacker following the above strategy can conclude, that if the application signals an error (or any other special behavior) on any future access, then it is more likely that this access pattern corresponds to an access to logical location 1 than to any other location.

In comparison, this error signaling is not problematic if the underlying protocol fulfills our stronger security goal. The reason is that the attacker can then only introduce failures that are equally likely for all logical locations and thus there is no bias in the correlation of the error signal and the access pattern.

Remark. It is important to note on what core property the above argument builds in order to develop schemes that are not susceptible to the same issue. The core property of the attack, which for example also explains why stronger ORAM notions, such as the next-read-pattern-hiding (NRPH) requirement in [CKW13], do not rule out the attack is the following: consider again equation (6.2). The problem here is that the expected level at which the newest version of a location resides is quite predictable and hence this element can be somewhat accurately deleted. Under the above attack, the access to this location has to throw an error, since the data is simply not available after deletion (it was chosen uniformly at random and is not redundantly stored, so the system would have to guess it which is not possible with very high probability). While enhancing the ORAM scheme to achieve NRPH improves resilience against rewinding attacks (in particular, rewinding twice has to yield the same pseudo-random walk), it does not improve on hiding where a logical element is stored approximately. In particular, the NRPH scheme of [CKW13] is also susceptible to the above attack.

6.5 Auditable Server-Memory Resources

In this section, we introduce the ideal abstraction of auditing mechanisms.

6.5.1 Basic, authenticated, and confidential auditable server memory

The ideal audit is described in Figure 6.17. It provides security guarantees only in a phase where an intruder is not active.[10] In this case, the check reveals whether the current memory blocks are indeed the newest version that the client wrote to the storage. If a single data block has changed, the ideal audit will detect this and output an error to the client. It is obvious that in case of a successful audit, this guarantee only holds up to the point where the server gains write-access to the storage again, in which case a new audit has to reveal whether modifications have been made. The goal of a scheme providing a proof of storage is to realize $\mathbf{SMR}_{\Sigma,n}^{k,\text{audit}}$, $\mathbf{aSMR}_{\Sigma,n}^{k,\text{audit}}$, or $\mathbf{cSMR}_{\Sigma,n}^{k,\text{audit}}$ from an ordinary server-memory resource.

6.5.2 Secure and auditable server memory

We present the ideal audit for secure memory resources in Figure 6.18. Due to the probabilistic nature of resource **sSMR**, the ideal retrievability guarantee for secure memory resources is a probabilistic one. Based on an additional parameter τ, the ideal audit of resource $\mathbf{sSMR}_{\Sigma,n}^{k,t_{\text{rep}},\tau,\mathsf{audit}}$ is successful if the probability that the entire memory cannot be retrieved is smaller than τ. The smaller τ, the stronger the retrievability guarantee.

6.6 Constructing Auditable Server-Memories

In this section, we provide constructions of auditable server-memory resources from ordinary server-memory resources. In order to show that a protocol achieves a construction, we again prove the conditions required by Definition 2.3.3. In this section, the default behavior at interface S is possibly more complicated, especially if interaction between the server and the client is required, for example if the client requests the server to compute a hash during an audit. Still, in the simpler case, the default behavior at interface S can be described by the usual dummy converter honSrv with the addition that it always inputs "allow" to the

[10] In fact, this is the only interesting case to consider, since if an intruder is active at the time of an audit, he can freely decide on the success or failure of the audit. We omit this second case in our specifications for simplicity.

Resource $\{\mathbf{a}, \mathbf{c}\}\mathbf{SMR}^{k,\mathsf{audit}}_{\Sigma,n}$

Server-memory resources are augmented with a new client capability as follows:

Interfaces $\mathtt{C}_r, r \in \{1, \ldots, k\}$

Input: $(\mathtt{read}, i) \in [n]$
Defined as in the respective resources **SMR**, **aSMR**, **cSMR**

Input: $(\mathtt{write}, i, x) \in [n] \times \Sigma$
Defined as in the respective resources **SMR**, **aSMR**, **cSMR**

Input: `audit` :
  if ACTIVE **and not** INTRUSION **then**
    output `auditReq` at $\mathtt{S}_H$
    Let $d \in \{\text{allow}, \text{abort}\}$ be the returned value from $\mathtt{S}_H$
    if $d =$ allow **then**
      $\mathbb{M}' \leftarrow$ empty table
      for $i = 1$ to n **do**
        if $\exists k, x, t : \textsc{Hist}[k] = (t, \mathtt{W}, i, x)$ **then**
          $k_0 \leftarrow \max\{k \mid \exists t, x : \textsc{Hist}[k] = (t, \mathtt{W}, i, x)\}$
          Parse $\textsc{Hist}[k_0]$ as $(t, \mathtt{W}, i, x_0)$
          $\mathbb{M}'[i] \leftarrow x_0$
        else
          $\mathbb{M}'[i] \leftarrow \lambda$
      if $\mathbb{M}' = \mathbb{M}$ **then**
        return `accept`
      else
        return `reject`
    else
      return `reject`

Figure 6.17: Description of the auditable server-memory resources (only difference to ordinary server memory shown).

Resource $\mathbf{sSMR}_{\Sigma,n}^{k,t_{\text{rep}},\tau,\mathsf{audit}}$

Secure server-memory resources are augmented with a new client capability as follows:

Interfaces $\mathtt{C}_r, r \in \{1,\ldots,k\}$

Input: $(\mathtt{read}, i) \in [n]$
Defined as for **sSMR**

Input: $(\mathtt{write}, i, x) \in [n] \times \Sigma$
Defined as for **sSMR**

Input: `audit` :
  if ACTIVE **and not** INTRUSION **then**
    output `auditReq` at $\mathtt{S}_H$
    Let $d \in \{\text{allow}, \text{abort}\}$ be the returned value from $\mathtt{S}_H$
    if $d =$ allow **then**
      $p \leftarrow 1 - \prod_{i=1}^{n}(1-\alpha^{t_{\text{rep}}-E_i})$, where $E_i := |\{j \mid j \in [t_{\text{rep}}] \text{ and } \mathbb{M}[i,j] = \epsilon\}|$
      (p is the probability that the value of at least one cell cannot be obtained based on the current state of the resource.)
      if $p \leq \tau$ **then**
        return `accept`
      else
        return `reject`
    else
      return `reject`

Figure 6.18: Description of the auditable secure server-memory resources (only the difference to the secure server memory is shown).

resource upon an audit request. We do not assign it a new name as it is clear from the context.

6.6.1 Making Authentic Server-Memory Resources Auditable

We now describe a straightforward way to achieve an auditable and authentic (or confidential) server-memory resource from an authentic (or confidential) server-memory resource. We again denote the storage content as $F = (F_1, \ldots, F_\ell)$, with $F_i \in \Sigma$. The main idea is to encode the entire storage F (for example an entire backup file) using an erasure code to tolerate a certain fraction of deleted symbols F_i. The audit consists

sampling a sufficient number of random positions of the encoded version of F and to correctly decide whether the information on the server is sufficient to decode the file. This straightforward idea has been studied before, for example in [SW13, JK07], and we briefly show how this idea is implemented in our model.

Assumed and constructed resource. The assumed resource is an authenticated server-memory resource of size ℓ' (which is used to store a single file consisting of ℓ blocks) and alphabet Σ. The system achieved is an auditable and authenticated server-memory resource with alphabet Σ and ℓ locations.

The protocol. We now describe the protocol in more detail by specifying the two client converters ecInit (for initialization) and $\mathsf{ecAudit}$ (to implement the audits). We note that the default server behavior for this section equals the dummy one which never deletes anything and allows all audit requests.

In the sequel, let (E, D) be an (ℓ', ℓ, d) erasure code as in Definition 2.2.5. On input $\texttt{init}$ to ecInit, the converter calls $\texttt{init}$ of its connected resource and computes the encoding $F' \leftarrow \mathsf{E}(\lambda^\ell) \in \Sigma^{\ell'}$, sets the counter ctr to 0. ctr can be seen as the version number or identifier of the currently stored memory content. Finally, the converter stores at each location $i \in [\ell']$ of $\mathbf{aSMR}^k_{\Sigma,\ell'}$ the pair $(F'_i, 0)$.

On $(\texttt{read}, i)$ to either ecInit or $\mathsf{ecAudit}$, the converter retrieves the whole memory content via $(\texttt{read}, i)$ requests and obtains for each cell either a pair (v_i, ctr') or the error symbol ϵ. If $ctr' \neq ctr$ (version mismatch) or if ϵ was returned, set $\bar{F}_i \leftarrow \perp$. Otherwise set $\bar{F}_i \leftarrow v_i$. If $|\{i \in [\ell'] \mid \bar{F}'_i = \perp\}| \geq d$, then output ϵ at the outer interface, otherwise, compute $F \leftarrow \mathsf{D}(\bar{F}')$, where $\bar{F}' = (\bar{F}'_1, \ldots, \bar{F}'_{\ell'})$, and output F_i.

On $(\texttt{write}, i, F'_i)$, where $F'_i \in \Sigma$, to either ecInit or $\mathsf{ecAudit}$, the converter first executes the same instructions as on input $(\texttt{read}, i)$ to retrieve the currently (outsourced) storage content F. If and only if the F is successfully retrieved, the converter increments ctr, updates the single location $F_i \leftarrow F'_i$ and re-encodes the new memory content F' as $\bar{F}' \leftarrow \mathsf{E}(F')$ and finally outputs $(\texttt{write}, i, (\bar{F'}_i, ctr))$ for all $i = 1 \ldots \ell'$.[11]

[11] If the code would additionally support local updates and local decoding then reading and writing could be implemented more efficiently.

Finally, on a query (`audit`) to converter $\mathsf{ecAudit}$, the converter chooses a random subset $S \subseteq [\ell']$ of size t and outputs $(\texttt{read}, i)$ to $\mathbf{aSMR}$ for each $i \in S$ to retrieve the memory content at that location. If and only if all read instructions for $i \in S$ returned a pair (and not ϵ) and the counter of all pairs are equal to the locally stored value ctr, then output `success`.

The security of this scheme follows from the following theorem.

Theorem 6.6.1. *Let $\ell, \ell', d \in \mathbb{N}$. Let (E, D) be an (ℓ', ℓ, d)-erasure-coding scheme for alphabet Σ and error symbol $\perp$ and let ρ be the minimum fraction of blocks needed to recover the file, i.e., let $\rho = 1 - \frac{d-1}{\ell'}$.* [12] *Then the above protocol $\mathsf{ecCheck} := (\mathsf{ecInit}, \mathsf{ecAudit}, \ldots, \mathsf{ecAudit})$ (with k copies of $\mathsf{ecAudit}$) that chooses a random subset of size t during the audit, constructs the auditable server-memory resource $\mathbf{aSMR}^{k,\mathsf{audit}}_{\Sigma^\ell,1}$ from the authentic server-memory resource $\mathbf{aSMR}^{k}_{\Sigma,\ell'}$, with respect to the simulator sim_{ec} (described in the proof of the theorem) and the pair $(\mathsf{honSrv}, \mathsf{honSrv})$. More specifically, for all distinguishers $\mathbf{D}$ performing at most q audits,*

$$\Delta^{\mathbf{D}}(\mathsf{honSrv}^{\mathsf{S}}\ \mathsf{ecCheck}_{\mathcal{P}}\ \mathbf{aSMR}^{k}_{\Sigma,\ell'}, \mathsf{honSrv}^{\mathsf{S}}\mathbf{aSMR}^{k,\mathsf{audit}}_{\Sigma,\ell}) = 0$$

$$\textit{and} \qquad \Delta^{\mathbf{D}}(\mathsf{ecCheck}_{\mathcal{P}}\ \mathbf{aSMR}^{k}_{\Sigma,\ell'}, \mathsf{sim}^{\mathsf{S}}_{ec}\mathbf{aSMR}^{k,\mathsf{audit}}_{\Sigma,\ell}) \leq q \cdot \rho^t.$$

Proof Sketch. Assume that a fraction α of cells of the real world authentic server memory have been deleted such that a $\beta := 1 - \alpha$ fraction is still available. A standard bound for binomial coefficients assures that the probability of selecting a subset of only good cells during an audit is $\frac{\binom{\beta \cdot m}{|S|}}{\binom{m}{|S|}} \leq \beta^{|S|}$. In the bad case where decoding would not be possible, i.e., if $\beta < \rho$, we see that for an arbitrary distinguisher $\mathbf{D}$ in the real setting, the probability that the audit succeeds is no larger than $\rho^{|S|}$.

We only prove the security condition and describe the simulator sim_{ec}. It internally maintains a simulated real-world view on the most recent memory content of size ℓ' by setting the memory content of simulated location i to ϵ on a $(\texttt{delete}, i)$ instruction from the distinguisher (and updating the memory content to the last value written in case of a $(\texttt{restore}, i)$ command. The simulator further maintains a simulated history, which is built based on the history of the ideal resource $\mathbf{aSMR}$ as follows: upon an audit request, the appropriate number of read-instructions are added to

[12] For example, for $\ell' > \ell$ and alphabet $\Sigma = \mathbb{F}_{q > \ell'}$ the systematic Reed-Solomon code over Σ has $d = \ell' - \ell + 1$ and thus $\rho = \frac{\ell}{\ell'}$.

the simulated history (for each read request to deleted location the entry $(t, \texttt{Fail}, i)$ is added). For each entry $(t, \texttt{R}, i)$ or $(t, \texttt{Fail}, i, F_i)$ in the ideal world history, the simulator replaces this entry by ℓ' read instructions in its own simulated history (for each read request to deleted location the entry $(t, \texttt{Fail}, i)$ is added). An entry $(t, \texttt{W}, i, F_i)$ (indicating a successful write operation) is replaced by ℓ' read-instructions (to all simulated locations) and ℓ' write instructions where each write instructions writes the pair $(\bar{F}_i, ctr)$ consisting of one symbol of the encoded and updated version of the memory content F together with the current counter ctr which is increased on each successful write operation.

If, after a deletion command, the number of simulated memory locations, that are equal to ϵ or associated with a too small counter, exceeds $d-1$, the simulator deletes all memory locations of the ideal resource. Similarly, after a restore-command (which restores the last valid value stored by the client at that location), if the number of invalid memory locations (including wrong counter values) drops below $d-1$, the simulator restores the entire memory content of the ideal resource.

On an audit request, the simulator simulates the random locations that are probed and evaluates if the test succeeds. If so, it allows the resource to output the right result to the client, and otherwise it instructs the resource to output `reject`. The simulation is perfect up to the point where the following event BAD happens: *An audit succeeds when more than $d-1$ locations are invalid.* The probability of distinguishing can hence be upper bounded by the probability that event BAD happens in an execution. To see this, in case $\neg$BAD, the whole (logical) memory content is intact as long as there are no more than $d-1$ deletions (or invalid counter values) to the real or simulated memory content. When the client initiates an audit, the simulator simulates the execution on its simulated memory which has the same distribution as in the real world and hence the probability to succeed is the same. In case of an unsuccessful (real or simulated) audit both, the ideal and the real system output `reject`. In case the check succeeds, both resources output `success` and the whole memory can be retrieved: either the simulator has not deleted the memory contents in this case, or, in the real system, less than $d-1$ locations are invalid such that decoding is successful. The statement follows. □

6.6.2 Making Secure Server-Memory Resources Auditable

We reduce the problem of auditing secure server-memory resources to the problem of estimating the corruption factor α. Each protocol chooses a tolerated threshold ρ and stores the data with replication factor t_{rep} that compensates data loss up to the corruption threshold ρ. To make sure that all values can be retrieved with a certain probability, the protocol tests t_{audit} fixed locations to estimate whether the parameter α has already reached the tolerated threshold ρ. In a first variant, the audit is successful if none of the probed locations return an error. In a second variant, we obtain similar results if the t_{audit} trials are used to obtain a sufficiently accurate estimate of α. The constructions are parameterized by the tolerated threshold ρ and by the desired retrievablity guarantee τ. The values of t_{audit} and t_{rep} depend on both of these parameters. The dependency is roughly as follows: The stronger the desired retrievability guarantee should be, the higher the value of t_{rep} needs to be. However, the smaller the value of the tolerated threshold ρ is, the smaller the value of t_{rep} can be. On the other hand, a smaller value of the threshold ρ implies a higher value of t_{audit}.

Assumed and constructed resource. The desired resource is an auditable secure server-memory resource of size ℓ and with retrievability guarantee τ. Recall that if an audit is successful, it means that the probability that any memory location is not accessible any more is smaller than τ. The assumed resource is a secure server-memory resource with replication t_{rep} and size $\ell + t_{\text{audit}}/t_{\text{rep}}$ whose values are determined below.

The protocol. As before, the protocol consists of the converters statInit (initialization), statAudit (client), and honest server behavior statSrvAudit. The server behavior is equal to the dummy behavior of the last section. So we only describe the protocol for the client. The protocol is parameterized by t_{audit}. For the sake of presentation, we do not explicitly write it as it is clear from the context. On input `init` to statInit, the converter calls `init` and sets FLAG $\leftarrow 0$. The variable FLAG records whether the protocol has ever detected an error when writing or reading to the server. If equal to one, it signals that misbehavior has been detected and will provoke subsequent audits to reject. The flag does not influence ordinary client

read and write requests. On $(\texttt{read}, i)$ to either $\mathsf{statInit}$ or $\mathsf{statAudit}$, the converter outputs $(\texttt{read}, i)$ to retrieve the value at memory location i or the error symbol ϵ, and outputs this returned value at its outer interface. In the case of an error, set $\text{FLAG} \leftarrow 1$. [13] On $(\texttt{write}, i, v)$ to either $\mathsf{statInit}$ or $\mathsf{statAudit}$, the converter outputs $(\texttt{write}, i, v)$ to write the value v at location i of the server. Again, if an error is observed, it sets $\text{FLAG} \leftarrow 1$. Finally, on input $\texttt{audit}$ to converter $\mathsf{statAudit}$, the converter immediately returns $\texttt{reject}$ if $\text{FLAG} = 1$. If $\text{FLAG} = 0$ the audit is executed as follows:[14], the converter issues t_{rep} read requests to each logical memory location $r = \ell+1, \ldots, \ell+\frac{t_{\text{audit}}}{t_{\text{rep}}}$. If and only if no read instruction returned the error symbol ϵ, then output $\texttt{success}$. Otherwise, the output is $\texttt{reject}$ and the flag is updated to $\text{FLAG} \leftarrow 1$. [15] The security of this scheme follows from the following theorem.

Theorem 6.6.2. *Let Σ be an alphabet, let $\ell, \kappa, t_{rep}, t_{audit}, d \in \mathbb{N}$ such that $d = \frac{t_{audit}}{t_{rep}}$, and let $\rho, \tau \in (0,1)$ such that*

$$t_{rep} > \frac{\log(\tau) - \log(\ell)}{\log(\rho)}, \qquad t_{audit} > \frac{-\kappa}{\log(1-\rho)}. \tag{6.3}$$

The above described protocol $\mathsf{statCheck} := (\mathsf{statInit}, \mathsf{statAudit}, \ldots, \mathsf{statAudit})$ *(with k copies of* $\mathsf{statAudit}$*) parameterized by t_{audit}, constructs the auditable secure server-memory resource $\mathbf{sSMR}^{k,t_{rep},\tau,\mathsf{audit}}_{\Sigma,\ell}$ from the secure server-memory resource $\mathbf{sSMR}^{k,t_{rep}}_{\Sigma,\ell+d}$ and a local memory (which stores the variable* FLAG*), with respect to the simulator sim_{stat} (described in the proof) and the pair* $(\mathsf{honSrv}, \mathsf{honSrv})$*. More specifically, for all distinguish-*

[13] One could relax this by introducing a tolerance *tol* and requiring that $\text{FLAG} \leftarrow 1$ only if more than *tol* of the t_{rep} copies of a memory location failed. Our results formalize zero tolerance and where t_{rep} is the minimal number required to obtain the desired retrievability guarantee.

[14] From a statistical point of view, if $\text{FLAG} = 0$, we have t_{audit} independent samples to estimate the parameter α.

[15] One could again introduce a tolerance *tol* and require that $\text{FLAG} \leftarrow 1$ only if more than *tol* samples returned an error. Our results formalize zero tolerance and t_{audit} is the minimal number of samples required to get a sufficiently accurate estimate of α.

ers $\mathbf{D}$ *performing at most* q *audits,*

$$\Delta^{\mathbf{D}}(\mathsf{honSrv}^{\mathsf{S}}\ \mathsf{statCheck}_{\mathcal{P}}[\mathbf{L}, \mathbf{sSMR}^{k,t_{rep}}_{\Sigma,\ell+d}], \mathsf{honSrv}^{\mathsf{S}}\mathbf{sSMR}^{k,t_{rep},\tau,\mathsf{audit}}_{\Sigma,\ell}) = 0$$

and

$$\Delta^{\mathbf{D}}(\mathsf{statCheck}_{\mathcal{P}}[\mathbf{L}, \mathbf{sSMR}^{k,t_{rep}}_{\Sigma,\ell+d}], \mathsf{sim}^{\mathsf{S}}_{stat}\mathbf{sSMR}^{k,t_{rep},\tau,\mathsf{audit}}_{\Sigma,\ell}) \leq q \cdot 2^{-\kappa}.$$

⌜ As a numerical example, let us assume we are given a secure memory that can store one tebibyte of data, with a certain replication factor t_{rep}, and where each element of Σ represents a block of size 16 kibibytes. This yields $\ell = 2^{26}$. For a security level $\kappa = 128$, $\tau = 2^{-32}$, and $\rho = 2^{-9}$ we would need a replication factor of $t_{\text{rep}} \approx 6$ and the total size of retrieved data during an audit is roughly 700 mebibytes. Different applications can adjust these parameters according to their preference in order to trade security, storage overhead, and access time. ⌟

Proof. We start by describing the simulator. sim_{stat} internally maintains a simulated history, which is identical to the history of the ideal resource but where for each audit request, the appropriate number of read-requests are added. It further maintains a value FLAG which is initially 0.

On input $(\texttt{pollute}, \alpha)$ and $(\texttt{reducePollution}, \delta)$ it forwards this query to the ideal resource $\mathbf{sSMR}^{k,t,\tau,\mathsf{audit}}_{\Sigma,\ell}$. On input `getHist` the simulator reads the history of the ideal resource and updates its simulated history appropriately and returns it to $\mathbf{D}$. If at any time, the simulated history contains an entry $(k, \texttt{Failed}$ for some k, then FLAG is set to 1.

On an audit-request, the simulator first updates its simulated history. Then, it replies abort to $\mathbf{sSMR}^{k,t,\tau,\mathsf{audit}}_{\Sigma,\ell}$ in case FLAG $= 1$. In case FLAG $= 0$, sim_{stat} simulates t_{audit} read accesses, such that each access fails with probability α. In case there are failures, sim_{stat} outputs abort to the ideal resource to provoke a reject and internally sets FLAG $\leftarrow 1$. It further adds the appropriate entries $(k, \texttt{Failed})$ and $(k, \texttt{Access})$ to its simulated history.

The remainder of the proof proceeds in two steps. First, we bound the probability that a simulated audit succeeds (if FLAG $= 0$), although α is larger than the threshold ρ, by $2^{-\kappa}$. A straightforward statistical argument shows that as long as $\alpha < \rho$ (and FLAG $= 0$), the probability that after the audit (and before the next intrusion phase) retrieving any cell would result in a failure is smaller than τ. Hence, the probability of an imperfect simulation is negligible in κ.

We first compute the probability that an audit is passed in case $\alpha > \rho$ and $\text{FLAG} = 0$. let $X = \sum_{i=1}^{t_{\text{audit}}} X_i$, where X_i are independently distributed according to $X_i \sim \mathsf{Bernoulli}(\alpha)$.

$$\Pr[X = 0] = (1-\alpha)^{t_{\text{audit}}} \leq (1-\rho)^{t_{\text{audit}}} \leq 2^{-\kappa},$$

where we used the assumption that $t_{\text{audit}} > \frac{-\kappa}{\log(1-\rho)}$. We conclude that except with negligible probability, the audit only succeeds if $\alpha < \rho$.

Assuming $\alpha < \rho$, we prove that the probability that any value is not recoverable (given that $\text{FLAG} = 0$) is smaller than τ for the choices in equation (6.3). Again, each read request would fail independently with probability α. Using Bernoulli's inequality, we get

$$1 - (1-\alpha^{t_{\text{rep}}})^{\ell} \leq 1 - (1-\rho^{t_{\text{rep}}})^{\ell} \leq 1 - (1 - \ell\rho^{t_{\text{rep}}}) = \ell \cdot \rho^{t_{\text{rep}}},$$

and by the theorem assumption $t_{\text{rep}} > \frac{\log(\tau)-\log(\ell)}{\log(\rho)}$, we immediately have

$$\ell \cdot \rho^{t_{\text{rep}}} \leq \ell \cdot \rho^{\frac{\log(\tau)-\log(\ell)}{\log(\rho)}} = \ell \cdot \rho^{\frac{\log(\tau)}{\log(\rho)}} \cdot \rho^{\frac{\log(-\ell)}{\log(\rho)}} = \ell \cdot \tau \cdot \ell^{-1} = \tau.$$

This concludes the proof. □

Auditing via a direct estimate of α. We can replace the audit of protocol $\mathsf{statCheck}$ by a direct estimation of the parameter α. In case that the estimation $\bar{\alpha}$ is sufficiently accurate, say up to $\frac{\rho}{2}$ with very high probability, verifying that $\bar{\alpha} < \frac{\rho}{2}$ is sufficient to obtain the desired retrievability guarantee and the audit returns `success`. The audit itself consists of obtaining t_{audit} independent samples via read-requests to the d extra locations of the secure server-memory resource. In case it is not possible to obtain that many samples, for example because certain locations failed during the last audit and would therefore output ϵ with probability 1 instead of α, the audit returns `reject`. If t_{audit} samples can be obtained, let n_e be the number of errors that occurred and define the estimate $\bar{\alpha} \leftarrow \frac{n_e}{t_{\text{audit}}}$. From the Chernoff-Hoeffding bound, we get that for arbitrary $\theta, \delta \in (0,1)$, if $t_{\text{audit}} > \frac{2+\theta}{\theta^2} \cdot \log(\frac{2}{\delta})$ independent samples of a Bernoulli distribution with parameter α are obtained, the probability that $\bar{\alpha} \in [\alpha - \theta, \alpha + \theta]$ is at least $1 - \delta$. Hence, setting $\theta < \frac{\rho}{2}$ and $\delta \leq 2^{-\kappa}$, we get an analogous result to the one above, but in general with higher values for t_{audit}.

On composing the previous modular steps. It is instructive to wrap up the results until this point: We have shown how to construct an auditable secure server-memory resource from a secure server-memory resource by estimating the failure probability α. In Section 6.4.3, we have shown how to construct a secure server-memory resource from an authentic and confidential server-memory resource, which itself can be constructed from an insecure server-memory as shown in Section 6.4.1 and Section 6.4.2. We can invoke the composition theorem of constructive cryptography to conclude that the composition of all protocols constructs an auditable secure server-memory resource from an insecure one (and local storage). The composed protocol has strong security guarantees and is robust against an arbitrary number of failures and is comparable to existing schemes in terms of access times to read and write single data blocks. However, the gained security comes at the price of a theoretically larger server-side memory consumption due to replication, and higher audit times.

6.6.3 Revisiting the Hash-Based Challenge-Response Approach

Our model allows to formalize the security guarantees of a very simple hash-based challenge-response protocol that is often given as an introductory example to proofs of retrievability, but, to the best of our knowledge, lacks a formal security statement in other models. In a nutshell, the retrievability test asks the server to deliver the correct hash value of the (current) storage content concatenated with a uniform random challenge provided (and precomputed) by the client. The intuitive security claim is that the server cannot have modified or deleted the content before answering the challenge. For the sake of concreteness, we consider the setting where one client stores a single file F (modeled as a sequence of bits in this paragraph) on an insecure server memory and would like to audit this file at a later point in time. We assume an (ideal) hash function, i.e., a random oracle, $\mathsf{H} : \{0,1\}^* \mapsto \{0,1\}^r$ following the notation of [BF11] and denote by $x||y$ the concatenation of bitstrings x and y.

Assumed and constructed resource. We assume a random oracle, $\mathsf{H} : \{0,1\}^* \mapsto \{0,1\}^r$, which is made available to the parties by means of

a resource **H** that has an interface for the client and one for the server: On input $(\texttt{eval}, x)$ at any of its interfaces **H** returns $\mathsf{H}(x)$ at the same interface. We further assume a small local storage and a communication channel between client and server, which we denote by **Ch**. Formally, we define system **Ch** here as a resource with the two interfaces $\texttt{C}_1$ and $\texttt{S}$, such that whatever is input at one interface is output at the other interface. Other formalizations of a point-to-point channel are of course possible but immaterial to the findings in this section. Last but not least, we assume an ordinary insecure memory resource $\mathbf{SMR}^{1}_{\Sigma,\ell+\kappa}$, where $\Sigma = \{0,1\}$ and κ being the size of the challenge c. The desired functionality we want to achieve is the auditable insecure memory resource $\mathbf{SMR}^{1,\mathsf{audit}}_{\Sigma,\ell}$.

The protocol. We now describe the protocol in more detail: As usual, we specify an initialization converter hashInit, a client converter hashAudit, and the protocol for the honest server behavior srvHash. On input `init` to hashInit, the converter simply calls `init` of its connected resource. On $(\texttt{write}, 1, F)$ to either hashInit or hashAudit, where F is an ℓ-bitstring, the converter writes F to the server storage. It then chooses a uniform random challenge $c \in \{0,1\}^\kappa$ and computes $y \leftarrow \mathsf{H}(F||c)$ and stores c and y in the local storage. On $(\texttt{read}, 1)$ to either hashInit or hashAudit, the converter retrieves the content of the memory and outputs the first ℓ bits of the received content. Finally, on a query (`audit`) to converter hashAudit, if there is a challenge stored in local memory, the protocol writes c to the server memory at locations $\ell + 1 \ldots \ell + \kappa$ and sends a notification `auditReq` to the server via the bidirectional channel. On receiving a response y' on that channel from the server, the client protocol outputs `success` if and only if $y = y'$. In any case, the challenge c is deleted from the local storage. The next audit is only possible after executing a new write-query.[16] Finally, the server protocol srvHash, upon receiving an audit-request, simply evaluates H on the current memory contents and sends the result to the client via the bidirectional channel.

Insecurity of the approach. Unfortunately, the security of the hash-based challenge-response protocol above does not follow solely based on the random oracle assumption. The proof of this follows closely the intuition

[16] We assume that the client protocol rejects an audit if no challenge is stored in local memory. Note that one could of course prepare more challenges.

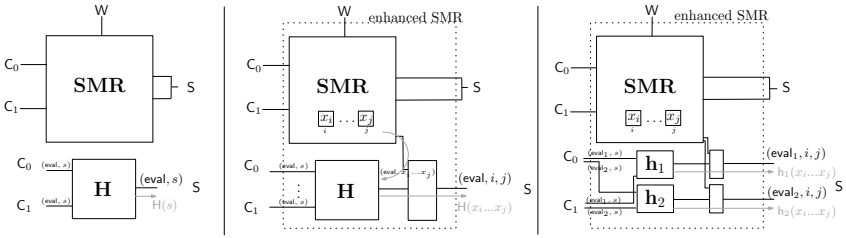


Figure 6.19: *Left:* Real system with unrestricted server access to the random oracle. The challenge-response protocol, executed in this setting, is not secure. *Center:* Real system with restricted server access. Under this stronger assumption, the challenge-response protocol is secure. *Right:* Real system with restricted access to two ideal compression functions. In this setting, the challenge-response protocol, where the hash is computed using a secure iterated construction like NMAC, is not secure in general.

that in a composable security framework, the environment "knows" the content of the server-memory resource. Hence, if the random oracle can be queried by the distinguisher on an arbitrary input, i.e., not restricted to the actual value stored in the server memory, it can always be queried on the correct input, irrespective of the actual content of the server-memory resource. This is formalized in the following lemma.

Lemma 6.6.3. *Let $\ell, \ell', \kappa, r \in \mathbb{N}$, with $\ell' = \ell + \kappa$, let $\Sigma := \{0,1\}$, and let $\mathbf{H}$ be a random oracle (with one interface for the client and one for the server). Then, the protocol above (specified by the converters* $\mathsf{hashInit}, \mathsf{hashAudit}, \mathsf{srvHash}$*) does not provide a secure proof of storage. More specifically, there is a distinguishing strategy such that for any simulator* sim *it holds that*

$$\Delta^{\mathbf{D}}(\mathsf{hashInit}^{\mathsf{C}_0}\mathsf{hashAudit}^{\mathsf{C}_1}[\mathbf{L}, \mathbf{Ch}, \mathbf{SMR}^1_{\Sigma,\ell'}, \mathbf{H}], \mathsf{sim}^{\mathsf{S}}\mathbf{SMR}^{1,\mathsf{audit}}_{\Sigma^\ell,1}) = 1.$$

Proof. To prove the statement, we describe the random experiment between a particular distinguisher $\mathbf{D}$ and the system $\mathbf{T}$, which either corresponds to the real system with the protocols attached, which is $\mathsf{hashInit}^{\mathsf{C}_0}\mathsf{hashAudit}^{\mathsf{C}_1}[\mathbf{L}, \mathbf{Ch}, \mathbf{SMR}^1_{\Sigma,\ell'}, \mathbf{H}]$ or to the ideal system with the simulator attached, i.e., $\mathsf{sim}^{\mathsf{S}}\mathbf{SMR}^{1,\mathsf{audit}}_{\Sigma^\ell,1}$. First, $\mathbf{D}$ inputs `init` at interface C_0 and queries, for some arbitrary file $F \neq 0^\ell$, $(\mathtt{write}, 1, F)$

and inputs `initComplete` at interface C_0. As the next step, **D** inputs `startWriteMode` at interface `W` and subsequently instructs the resource to delete the file by storing the all-zero string via queries $(\texttt{write}, i, 0)$ for all locations $i \in [\ell]$ at interface S_I and finally inputs `stopWriteMode` at interface `W`. **D** then inputs `audit` at interface C_1 to receive a challenge c.[17] **D** then queries **H** on input $F||c$ to receive the value y_0 and sends y_0 back to the client. As the last step the client retrieves the actual storage content by querying $(\texttt{read}, 1)$ at interface C_1. Let the returned file be F'. Finally, the distinguisher outputs 1 if and only if the audit is successful and F' is the all-zero bitstring. It is obvious that if **D** is querying the real system (with the protocol), then its output is 1 with certainty. However, in the ideal system, if the server memory content when the audit start is $F' \neq F$, then the ideal audit, by definition cannot be successful, irrespective of the simulator's actions. The distinguisher outputs 1 with probability zero in that case. The statement follows. □

Security under stronger assumptions. In this paragraph, we show that the additional assumption we have to make in order for the scheme to be secure, is to restrict adversarial random oracle evaluations by allowing inputs from the server storage only. In particular, the server is only allowed to query the random oracle via calls $(\texttt{eval}, i, j)$, $i \leq j$, and to obtain the hash value $\mathsf{H}(\mathbb{M}[i]|| \ldots ||\mathbb{M}[j])$ as opposed to receiving hash values for arbitrary bitstrings. See also Figure 6.19.

To turn this intuition into a formal statement, we consider the following functionality $\mathbf{SMR}^k_{\mathsf{H},\Sigma,\ell}$ which basically behaves like $\mathbf{SMR}^k_{\Sigma,\ell}$, but with two additional capabilities: Each client interface can, aside of ordinary read- and write-request, query $(\texttt{eval}, x)$ upon which the resource provides $\mathsf{H}(x)$ as output. Second, the server gets access to the random oracle via its interface, and is restricted to submit queries of the form $(\texttt{eval}, i, j)$ with $i \leq j$, and the resource returns the result of $\mathsf{H}(\mathbb{M}[i]|| \ldots ||\mathbb{M}[j])$ to the server. We prove the following theorem.

Theorem 6.6.4. *Let $\ell, \ell', \kappa, n \in \mathbb{N}$, with $\ell' = \ell + \kappa$, let $\Sigma := \{0,1\}$, and let H denote a hash function modeled as a random oracle. The above described protocol* $(\mathsf{hashInit}, \mathsf{hashAudit})$ *constructs the auditable server-memory resource* $\mathbf{SMR}^{1,\mathsf{audit}}_{\Sigma^\ell,1}$ *from the server-memory resource* $\mathbf{SMR}^1_{\mathsf{H},\Sigma,\ell'}$*,*

[17] We assume that in both worlds a challenge is output as otherwise distinguishing is trivial.

a local memory (of constant size), and a channel, with respect to the simulator sim_{hash} *(described in the proof) and the pair* $(\mathsf{srvHash}, \mathsf{honSrv})$. *More specifically, for all distinguishers* $\mathbf{D}$ *asking at most* q *queries*

$$\Delta^{\mathbf{D}}(\mathsf{hashInit}^{\mathsf{C}_0}\mathsf{hashAudit}^{\mathsf{C}_1}\mathsf{srvHash}^{\mathsf{S}}[\mathbf{L}, \mathbf{Ch}, \mathbf{SMR}^{1}_{\mathsf{H},\Sigma,\ell'}], \mathsf{honSrv}^{\mathsf{S}}\mathbf{SMR}^{1,\mathsf{audit}}_{\Sigma^\ell,1}) = 0$$

and

$$\Delta^{\mathbf{D}}(\mathsf{hashInit}^{\mathsf{C}_0}\mathsf{hashAudit}^{\mathsf{C}_1}[\mathbf{L}, \mathbf{Ch}, \mathbf{SMR}^{1}_{\mathsf{H},\Sigma,\ell'}], \mathsf{sim}^{\mathsf{S}}_{hash}\mathbf{SMR}^{1,\mathsf{audit}}_{\Sigma^\ell,1}) \leq q \cdot 2^{-\kappa} + 2^{-r}.$$

Proof Sketch. Since it is obvious that when the server is honest, the audit succeeds, we directly proceed to prove the security of the construction. We first describe the straightforward simulation. On any query by the distinguisher to read or write directly into the storage via server interface $\mathtt{S}_I$, the simulator simply forwards this request to $\mathbf{SMR}^{1,\mathsf{audit}}_{\Sigma^\ell,1}$. If the distinguisher inputs the query $(\mathtt{eval}, i, j)$ for $1 \leq i \leq j \leq \ell + \kappa$, the simulator computes the string $s \leftarrow \mathbb{M}'[i]||\ldots||\mathbb{M}'[j]$, and if there is no internally stored pair (s, y) for this string, choose $y \twoheadleftarrow \{0,1\}^r$ and store (s, y) internally for future reference. Finally, output y as the (simulated) random oracle output to the distinguisher.

Last but not least, when the client starts the first audit for the most recent uploaded file (note that by definition such a requests occurs only in a phase where no intruder is active), the simulator internally chooses a challenge $c \twoheadleftarrow \{0,1\}^\kappa$. Then, the simulator retrieves the history of the resource to check which file F was written to the storage and defines $s \leftarrow F||c_1||\ldots||c_\kappa$ and checks whether there is a recorded pair (s, y). Only if none is recorded, chooses $y_0 \twoheadleftarrow \{0,1\}^r$ and store the pair (s, y_0). Finally, sim_{hash} stores c at locations $\mathbb{M}'[\ell+1]$ to $\mathbb{M}'[\ell+\kappa]$ of its simulated memory. After this, it outputs the notification `auditReq` to the distingusiher (as coming from the bidirectional channel). If the distinguisher's response to this audit request equals y_0, then the simulator outputs allow and otherwise output abort to $\mathbf{SMR}^{1,\mathsf{audit}}_{\Sigma^\ell,1}$.

We now consider the an execution of a distinguisher with either the real system or the ideal system. On an audit-request by the client, which happens only in a phase where the distinguisher is not allowed to write to the server-memory resource, the client reveals the challenge c by writing it into the server storage. Let us denote the current server storage at this

point as R. We observe that the server is only capable of evaluating the random oracle on $R||c$ or on $F||c$ by restoring the original memory cell. Hence, assume that the distinguisher does not restore and the memory content is $R \neq F$. Hence, in both worlds, the distinguisher does not learn receive the value $y_0 = \mathsf{H}(F||c)$ at this point. In particular, the probability of guessing the correct hash output given that $\mathbf{D}$ has never evaluated the random oracle on $s = F||c$ is 2^{-r}. Furthermore, the probability that the distinguisher has ever evaluated the random oracle on $F||c$ before the audit was initiated, is no larger than $q \cdot 2^{-\kappa}$. We observe that if none of these two events occur, then the real and ideal systems behave identically. Indeed, a retrievability check is passed in both worlds if and only if the memory content is the original file F and the distinguisher sends the correct hash value $y_0 = \mathsf{H}(F||c)$ to the client. The statement follows. □

On replacing the monolithic random oracle. We consider iterated constructions of random oracles $\mathsf{H} : \{0,1\}^* \mapsto \{0,1\}^r$ from ideal compression functions $\mathsf{h}_1 : \{0,1\}^\kappa \times \{0,1\}^n \mapsto \{0,1\}^n$ and $\mathsf{h}_2 : \{0,1\}^n \mapsto \{0,1\}^r$ $(n, r, \kappa > 0)$ from [CDMP05]. We formally show that in our setting, the fact that an iterated construction realizes a random oracle does not imply that the iterated construction realizes resource $\mathbf{SMR}^1_{\mathsf{H},\Sigma,\ell'}$ from resource $\mathbf{SMR}^1_{\mathsf{h}_1,\mathsf{h}_2,\Sigma,\ell}$. In other words, even if the server access to the functions is restricted as required above by Theorem 6.6.4 and illustrated in Figure 6.19, their applicability is not generally safe in the context of audits. This observation meets our intuition and has already been observed in [RSS11]. The intuitive reason why it fails is that certain constructions (like NMAC) allow the server to compute a result in multiple stages, such that he can store an intermediate result, ignore the original memory, and still compute the correct hash value.

For simplicity, we focus on the NMAC construction that was shown to securely realize a random oracle H from two ideal compression functions h_1 and h_2 [CDMP05]. The input is a file $F = (F_1||F_2||\ldots||F_\ell)$ of $\ell = \kappa \cdot l$ bits, let us denote the ith block (having κ bits) as F^i. Let $y_0 \leftarrow 0^n$ be the initial block.[18]. Compute for each block i from 1 to l, $y_i \leftarrow \mathsf{h}_1(F^i, y_{i-1})$. Finally, compute and return $Y \leftarrow \mathsf{h}_2(y_l)$ as $\mathsf{H}(F)$. Let nmac_{cli} be the client converter that simply relays all queries and responses not concerning

[18]We assume this initial value to be prepended to F such that the computation formally gets $F' = 0^n||F$ as the only input. This is only a syntactic simplification

the random oracle evaluations and on input $(\texttt{eval}, x)$ at its outer interface, computes the hash value according to the NMAC construction above. Similarly, the honest server converter nmac_{srv} simply relays all queries and responses not concerning the random oracle evaluations and on input $(\texttt{eval}, i, j)$ at the outer interface evaluates the NMAC construction using the appropriate instructions to the resource.

Lemma 6.6.5. *Let $\ell, n, r, \kappa > 0$ be integers such that $\ell > n + \kappa$, and let $\Sigma := \{0,1\}$. Let H denote a random oracle and h_1 and h_2 be ideal compression functions as introduced above. Then, the client protocol $(\mathsf{nmac}_{cli}, \mathsf{nmac}_{cli})$ described above does not construct the system $\mathbf{SMR}^1_{\mathsf{H},\Sigma,\ell}$ from system $\mathbf{SMR}^1_{\mathsf{h}_1,\mathsf{h}_2,\Sigma,\ell}$ if the server is possibly dishonest. In particular, there is a distinguishing strategy such that for any simulator sim, making at most q random oracle queries, it holds that*

$$\Delta^{\mathbf{D}}(\mathsf{nmac}^{\mathsf{C}_0}\mathsf{nmac}^{\mathsf{C}_1}\mathbf{SMR}^1_{\mathsf{h}_1,\mathsf{h}_2,\Sigma,\ell}, \mathsf{sim}^{\mathsf{S}}\mathbf{SMR}^1_{\mathsf{H},\Sigma,\ell}) \geq 1 - q \cdot 2^{-\kappa} - 2^{-r}.$$

Proof Sketch. We describe a distinguisher $\mathbf{D}$ that interacts either with the ideal world $\mathsf{sim}^{\mathsf{S}}\mathbf{SMR}_{\mathsf{H}}$ or with the real world $\mathbf{SMR}^1_{\mathsf{h}_1,\mathsf{h}_2,\Sigma,\ell}$. The distinguisher first chooses a uniform random bitstring s of length $\ell - 1$ and stores $F := s||1$ in the memory and pre-computes $y := \mathsf{H}(F, c)$ for a uniformly random challenge c (of length κ) via an input $(\texttt{eval}, F||c)$ at interface $\mathtt{C}_1$. A second step, $\mathbf{D}$ performs the "first stage" of the NMAC computation using the interface $\mathtt{S}$: having obtained y_{i-1}, $\mathbf{D}$ writes this value back to an appropriate location, say j, of the server storage and queries $(\texttt{eval}_1, j, j + \kappa + n)$ to receive the intermediate value y_i and proceeds to until obtaining y_ℓ. Finally, $\mathbf{D}$ sets the server memory to $y_l||0^{\ell-n}$ via a write-command at the server interface and then issues `stopWriteMode` at interface $\mathtt{W}$ which disallows adversarial write-access at interface $\mathtt{S}$. Next, $\mathbf{D}$ writes the challenge c to the server storage via the client interface $\mathtt{C}_1$ and completes the evaluation of NMAC by computing $y' \leftarrow \mathsf{h}_2(\mathsf{h}_1(1, \ldots, n||c))$ by appropriate evaluation queries at the server interface. To decide on its output bit, $\mathbf{D}$ reads the current content R of the memory at interface $\mathtt{C}$ and decides on 1 if and only if $R = y_l||c||0^{\ell-n-\kappa}$ and $y = y'$.

If $\mathbf{D}$ is connected to the real system then, by design of the experiment, the probability that $y = y'$ and $R \neq F$ is 1. The reason is that the memory content $y_l||c||0^{\ell-n-\kappa}$, is sufficient to compute the correct hash value $\mathsf{H}(F||c)$.

If **D** is connected to the ideal system, the probability that $y = y'$ and $R \neq F$ is significantly smaller and is based on the observation that the simulator can only compute $H(F||c)$ with non-negligible probability if it can evaluate the storage its random oracle on input $F||c$ which has to reside in memory. By the time the simulator learns c, he has already lost his write-access to the resource. Hence, the probability of $y = y'$ given that the storage content in the end of the experiment is $R \neq F||c$ is upper bounded by $q \cdot 2^{-\kappa} + \cdot 2^{-r}$. □

Part III

Digital Signatures

Chapter 7

A Constructive Model for Signatures

7.1 Introduction

Digital signature schemes are a fundamental building block in numerous applications. This makes it particularly difficult to define an idealization that satisfies both, capturing the unique properties of the scheme and nevertheless achieving a good level of abstraction. This chapter is devoted to a proposal of such a model. The methodology we put forward here is not restricted to digital signature schemes and can in principle be applied to any low-level primitive if desired.

7.1.1 Motivation

A digital signature scheme (DSS) allows a signer to authenticate a message such that everyone can verify the authenticity. The signer initially generates an asymmetric key pair consisting of a *signing key* and a *verification key*. The signing key, which is kept secret by the signer, allows to generate signatures for messages. The verification key is made public and allows to verify that a message was indeed signed using the corresponding signing key. DSSs are a crucial component in many of today's widely-used cryptographic protocols. They underlie the public-key infrastructure (PKI) that

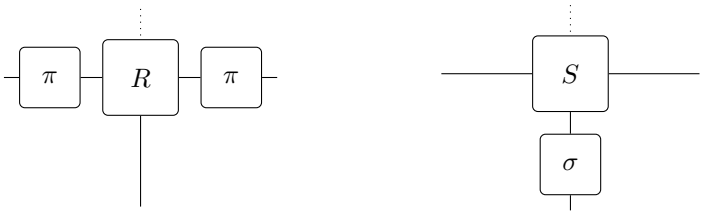


Figure 7.1: Recalling the construction notion for the discussion: **Left:** Execution of protocol π in the real-world model. **Right:** Ideal-world model described by S with simulator σ. In both figures, the dotted lines are "free" interfaces explained below.

is used to provide authentication in most Internet protocols, and they are used to authenticate e-mails as well as to provide non-repudiation for electronic documents. They are also used as a building block in numerous cryptographic protocols.

7.1.2 Methodology and Outline of the Model

We outline the most important aspects of the model which appears in [BMT18].

7.1.3 Formalizing Message Authentication

The core idea of our approach is that digitally signing a message can be understood as the signer's declaration that the message belongs to a certain context, which is described by the verification key. This context may be the signer's commitment to be legally liable for the content of the message (e.g., a contract), or simply that the message is meant to originate from the signer. Abstractly, this can be understood as writing the message to a certain type of *repository* that allows other parties to verify for given messages whether they have been written to the repository, i.e., assigned to the context.

Signature schemes as constructions. We formalize the security of a DSS in the real-world/ideal-world paradigm and based on different

types of repositories to which messages can be written and from which messages can be read, by different parties with potentially different access permissions. As described above, the goal of using the signature scheme in the described way can be seen as constructing an *authenticated* repository, where only the signer can write messages and all verifiers can check the validity. This repository takes the role of the ideal system that should be achieved.

Using a signature scheme requires an authenticated repository that can hold one message. This repository is used to transmit the signature verification key. We also assume one repository that can hold multiple messages, but this repository can be *insecure*, meaning that write access to the repository is not exclusive to the signer. This repository is used to transmit the signature strings. We also make the storage of the signing key explicit as a *secure* repository where both write and read access is exclusive to the signer. These three assumed repositories correspond to R in Figure 7.1.

A signature scheme then uses the described repositories in the obvious way: the signer begins by generating a key pair, writes the signing key to the secure repository and the verification key to the authenticated one. Upon a request to sign a message m, the signer retrieves the signing key from the secure repository, computes the signature, and stores it in the insecure repository. For checking the validity of a message m, a verifier reads the verification key from the authenticated repository and the signature from the insecure one, and runs the signature verification algorithm. Our security statement is, then, that this use of the signature scheme constructs the desired authenticated repository for multiple messages from the three described assumed repositories.

The advantage of such a composable security statement is that applications and higher-level protocols can be designed and analyzed using the abstraction of such a repository; in particular, no reduction proof is required since the composition theorem immediately guarantees the soundness of this approach. More technically, if a protocol π constructs S from R and protocol π' constructs T from S, then composing the two protocols leads to a construction of T from R.

Abstract communication semantics. The purpose of a repository is to model the fact that a certain message written by one party can be

made accessible to a different party in an abstract manner. Indeed, a DSS is a generic security mechanism and can be used by various applications; the definition of a DSS should abstract from the particular way in which the verification key and the signature are delivered to the verifier. For instance, a communication network used for transmission may guarantee messages to be delivered within a certain time, or an attacker may be able to eavesdrop on messages. Using a DSS—intuitively—preserves such properties of the communication network. The repositories used in this work are general enough to model various different such concrete types of transferring the values.

This generality is, more technically, achieved through a *free* interface that appears in both the real-world and the ideal-world model. In the random experiment, this interface is accessed directly by the distinguisher. The free interface is reminiscent of the environment access to the global setup functionality in the GUC model [CDPW07a], but in our model each resource/functionality can have such a free interface.[1]

A free interface allows the distinguisher to interact with both the assumed resource R and the constructed resource S directly, indicated by the dotted lines in Figure 7.1 This results in a stronger and more general condition compared to considering the capabilities at that interface as part of the attacker's interface and, therefore, in the ideal-world model providing them to the simulator. More intuitively, the free interface can be seen as a way for the distinguisher to enforce that certain aspects in the real and the ideal world are the same. We will use the free interface to let the distinguisher control the transmission semantics; this leaves our statements general and independent of any concrete such semantics.

In more detail, the write and read interfaces of the repository are defined to write to or read from buffers associated to the interface. The repository also has free interfaces that control the transfer of messages from write buffers to read buffers. In other words, capabilities such as writing messages to a buffer in the repository or reading messages from one are separated from the mechanisms for making messages written to the repository visible at a specific reader interface. Control over the operations governing the visibility is granted to the environment—this makes the security statements independent of specific network models.

[1] The direct communication between the environment and the functionality requires a modification of the control function in UC, but does not affect the composition theorem. In most formal frameworks [PW01, KT08, MR11], no modification is necessary.

In particular, the statements imply those in which these capabilities are granted to an attacker controlling the network.

Interfaces and partitioning of capabilities. In our proposed model, we take the viewpoint that interfaces of a resource represent capabilities. Often, each interface can be seen as corresponding to one particular party in a given application scenario, which can then attach a protocol machine to this interface, as in Figure 7.1. Yet, for a general security definition such as that of a DSS we do not want to fix the number of possible verifiers in advance, or even prohibit that the signing key may be transmitted securely between and used by different parties. As one can always merge several interfaces and provide them to the same party, it is beneficial here to target a fine-grained partitioning of capabilities into interfaces, and therefore a fine-grained partitioning of the protocol into individual protocol machines.

For our repositories, this means that if each interface gives access to one basic operation (such as writing or reading one value), one can always subsume a certain subset of these capabilities into one interface and assign it to a single party. We achieve the most fine-grained partitioning by modeling each invocation of an algorithm of the signature scheme as a single protocol machine, and capture passing values between the machines explicitly via repositories.

Specifications. For generality or conciseness of description, it is often desirable to not fully specify a resource or functionality. For instance, a complete description of the construction would entail the behavior of the signature scheme in the case where a signature shall be verified before the verification key is delivered to the verifier. The approach generally used in the literature on UC in such cases is to delegate such details to the adversary, to model the worst possible behavior. In this work, we follow a more direct approach, and explicitly leave the behavior undefined in such cases.

Our formalization follows the concept of *specifications* by Maurer and Renner [MR16] and introduced in Section 2.3.6, which are sets of resources that, for example, fulfill a certain property. As such they are suitable to express an incomplete description of a resource, namely by considering the set of all resources that adhere to such a (partially defined) description.

Maurer and Renner describe concrete types of specifications such as all resources that can be distinguished from a specific one by at most a certain advantage, or all resources that are obtained from a specific one by applying certain transformations.

As explained in Section 2.3.6, we use specifications in this chapter to describe the behavior of a resource in environments that use the resource in a restricted way, in the sense that the inputs given to the resource satisfy certain conditions, such as that the verification key must have been delivered before messages can be verified. This alleviates the requirement of specifying the behavior of the resource for input patterns that do not occur in applications, and simplifies the description. Needless to say, this also means that for each application one has to show that the use of the resource indeed adheres to the specified conditions.

The repositories of our model. In summary, we consider specifications of repositories as described above. Repositories provide multiple interfaces, each of which allows exactly one write or read operation. A repository that allows for k write operations has k writer interfaces, and for n read operations it has n reader interfaces, and each operation can be understood as writing to or reading from one specific buffer. A write interface may allow the writer to input an arbitrary value from the message space, or, in a weaker form, it may allow the writer to only copy values from buffers at some read interfaces. A read interface may either allow to retrieve the contents of the corresponding buffer, or to input a value and check for equality with the one in the buffer.

The resource additionally provides free interfaces for transferring the contents of write buffers to read buffers. As discussed above, the access to these interfaces for managing the visibility of messages is given to the distinguisher, not the attacker, to abstract from specific communication semantics.

All repositories in this chapter can be viewed as specific instances of the one described above, where different types of capabilities are provided at different parties' interfaces. It is this assignment of capabilities to parties that decide on the type of the repository. For instance, a repository in which the attacker has only read-interfaces, but cannot write chosen messages, can be called *authenticated*, since all messages must originate from the intended writers. A repository where the attacker can also write

can be considered as *insecure*, since messages obtained by honest readers could originate either from honest writers or the attacker.

7.1.4 Relation to Previous Work

The concept of a DSS was first envisioned by Diffie and Hellman and referred to as *one-way authentication* [DH76]. Early instantiations of this concept were given by Rivest, Shamir, and Adleman [RSA78] and by Lamport [Lam79]. The provable-security treatment of DSS was initiated by Goldwasser, Micali, and Rivest [GMR88], who also introduced the first and still widely-used security definition called *existential unforgeability under chosen-message attack*. In this definition, a hypothetical attacker that has access to honestly computed signatures on messages of his own choice aims at creating a signature for some new message. A scheme is deemed secure if no efficient attacker can provide such a forgery with non-negligible probability.

Canetti [Can01a] and independently Pfitzmann and Waidner [PW01] developed security frameworks that allow for security-preserving composition of cryptographic schemes. In these frameworks, the security of a cryptographic scheme, such as a DSS, is modeled by idealizing the algorithms and their security properties, and a concrete scheme is then proved to satisfy the idealization under certain computational assumptions. Higher-level schemes and protocols that make use of a DSS can be analyzed using the idealized version of the scheme. One main advantage of composable frameworks is that they guarantee the soundness of this approach; a higher-level protocol proven secure with respect to an idealized signature scheme will retain its security even if the idealized scheme is replaced by any concrete scheme that is proven secure. In contrast to standard reductionist proofs, this method does *not* require to prove an explicit reduction from breaking the signature scheme to breaking the higher-level protocol; this follows generically from the composition theorem. Still, even in protocol analyses within composable frameworks, existential unforgeability remains widely used, despite the existence of composable models within these formal frameworks.

The first composable notion for digital signatures has been proposed by Canetti [Can01a, Can04] via an ideal signing functionality $\mathcal{F}_{\text{SIG}}$. The functionality idealizes the process of binding a message m to a public key vk via an ideal signature string s. In a nutshell, when the honest

sender signs a message, he receives an idealized signature string. This signature string allows any party to verify that the message has indeed been signed by the signer. $\mathcal{F}_{\text{SIG}}$ enforces consistency and unforgeability in an ideal manner: if the honest signer has never signed a message m, no signature string leads to successful verification. Likewise, verification with a legitimately generated signature string for a message m always succeeds. Special care has to be taken in case the signer is dishonest, in which case the above guarantees for unforgeability are generally lost. The formalization given by Backes, Pfitzmann, and Waidner [BPW03] in their framework follows a by and large similar approach.

Several approaches to defining signature idealizations have been proposed and different versions of the signature functionality have been suggested in previous work [Can01a, CR03, BH03, Can04, CSV16a, Pat05, GKZ10]. All these versions, however, require interaction with the ideal-model adversary for operations that correspond to local computations in any real-world scheme, such as the initial creation of the key pair (including [Pat05]) or the generation of a signature string (including [Can04]).

Camenisch *et al.* [CEK+16] point out that this unnatural weakness, allowing the adversary to delay operations in the idealized security guarantee, has often gone unnoticed and even lead to flaws in proofs of higher-level schemes based on signatures. As a further example, consider a signer S that has never signed a message m. If an honest party P verifies m with respect to some signature string s, the verification should fail. Yet, the adversary gets activated during any local verification request and can corrupt the signer just before providing the response. The adversary thus has complete freedom on whether to let P accept or reject the signature string s on message m. This behavior is arguably counter-intuitive and it is a property that signature schemes do not possess. The solution of Camenisch *et al.* [CEK+16] requires to modify the universal composability framework by introducing the concept of *responsive* environments and adversaries that are mandated to answer specific requests immediately to model local tasks. While Camenisch *et al.*do re-prove the composition theorem for their modified framework, such a modification of the framework has the downside of further increasing its complexity and, at least in principle, making security analyses in the original and modified frameworks incompatible.

Besides the technical difficulties in defining the signature functionality $\mathcal{F}_{\text{SIG}}$ consistently, it is less abstract than what one would expect, since

the signature string and the verification key are an explicit part of the interface. Indeed, Canetti [Can04, page 5] writes:

> The present formalization of $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{CERT}}$ is attractive in that it allows a very modular approach where each instance of the ideal functionality handles only a single instance of a signature scheme (i.e., a single pair of signature and verification keys). This has several advantages as described in this work. However, the interface of the signature scheme is somewhat less abstract than we may have wanted. Specifically, the interface contains an idealized "signature string" that is passed around among parties [...].

Indeed, Canetti [Can04, page 7] starts by describing a "first attempt" functionality $\mathcal{F}_1$ that is a "depository of signed messages," where the signer can input a message and the verifiers can check. This functionality can be seen as a simplified version of the authenticated repository we described above. He then argues, however, that including the technical details in the functionality's interface is inevitable, see [Can04, page 7]:

> The lack of explicit signature strings also causes some other modeling problems. For instance, modeling natural operations such as sending an "encrypted signature" that is usable only by the holders of the decryption key cannot be done in a modular way [...] We conclude that in order to capture our intuitive notion of signature schemes, an ideal signature functionality should make the "signature string" part of its interface. [...]

We want to argue here that, despite the similarity, the arguments given in [Can04] do not apply to our definition. The first argument is that the formulation binds the messages to the signer's identity instead of the verification key, which requires prior communication to transmit the verification key. While this argument is correct, and our definition makes the repository for transmitting the verification key explicit, we stress that the repositories abstract from concrete types of communication and merely specify that the correct verification key generated by the signer is accessible, in some way, to the verifier. The means of *how* it became accessible do not have to be specified.

The second argument is that (beyond requiring the communication of the signature string, which is analogous to the verification key), protocols that communicate a signature over a different connection than specified, such as an encrypted one, is a modeling challenge. One such protocol is SAML [HCH+15], where a signed assertion on the identity of a party is sent through a TLS connection. Despite the fact that this assertion is indeed encrypted, and SAML would therefore appear to be in the class of

protocols referred to by Canetti, it is shown in [BMT18] that our model, which does not explicitly expose the signature string, indeed allows to analyze the security of protocols like SAML. The reason is again that our model abstracts from the concrete communication semantics and in particular also allows to model the case where a signature is transferred securely.

There are protocols that make explicit use of the verification key or signature as a bit string and for which our model in its current form does not support a modular analysis. One example is the transformation from CPA-secure public-key encryption (PKE) to non-malleable PKE by Choi *et al.* [CDSMW08], where each ciphertext is protected via an instance of a one-time signature scheme, and the bits of the verification key are used to select a certain subset of instances of the CPA-secure PKE. For the security reduction to succeed, however, it is necessary that the verification key be not only a bit string, but that it also be different for each instance, with high probability. While this property is clearly satisfied by every secure DSS, and therefore also each DSS that realizes $\mathcal{F}_{\text{SIG}}$, it is not captured in the functionality alone, where the adversary can freely choose the verification key. Hence, a composable analysis of the Choi *et al.*scheme in the $\mathcal{F}_{\text{SIG}}$-hybrid model is inherently impossible. In summary, this shows that the property of outputting *some* string as the verification key is not sufficient at least for the application of [CDSMW08]. Another example are protocols that require parties to provide proofs (e.g., of knowledge) over inputs and outputs of the DSS algorithms. Yet, also here, the same issues appear with the formalization $\mathcal{F}_{\text{SIG}}$ that is independent of any concrete scheme. In summary, it remains open whether there is a natural scheme that can be modularly proved based on $\mathcal{F}_{\text{SIG}}$, but not using the more abstract definition we put forth in this paper.

Finally, our work can be seen as orthogonal to the work of Canetti et al. [CSV16a], which extends the model of Canetti [Can01a, Can04] to the case where verification keys are available globally. While our model does not restrict the use of the constructed resource, the central aspect of our work is the different paradigm underlying the specification of the functionalities.

7.1.5 Specific Contribution

The main contribution of [BMT18] on which this chapter is based is the formal model sketched in Section 7.1.3 above, which we formally specify in Section 7.2. We additionally prove several statements about DSSs using this model.

Capturing the security of a DSS. We define, in Section 7.3.1, the security of a DSS as constructing an authenticated repository, shown on the right-hand side of Figure 7.2, from an insecure repository, called "insecure **Rep**" on the left-hand side of Figure 7.2, an "authenticated **Rep**" to which one message can be written, and a "secure **Rep**" that allows to write a single message, but to which the adversary has neither read- nor write-interfaces. As shown in Figure 7.2, using the signature scheme, which consists of the systems labeled setup, write, and check, requires the two single-message repositories for distributing the signing and verification keys. In more detail, in write each message is signed and the signature input into the insecure repository. Checking whether a given message m has been written to the repository is done by verifying the received signature for m within check.

We then prove that this construction statement is equivalent to the existential unforgeability of secure digital signature schemes in the sense of [GMR88]:

Theorem (informal). *A DSS constructs an authenticated multi-message repository from an insecure multi-message repository, an authenticated single-message repository and a secure single-message repository if and only if it is existentially unforgeable.*

Following the discussion in [Can04], we have to argue that our abstract formalization of a signature scheme indeed models the intuitively expected properties of such a scheme. In particular, in Section 7.3.5, we show that the formalization directly models the *transferability* property of signature schemes in the sense that a receiver of a signature can forward it to another party, who can also verify it. For further applications we refer to [BMT18].

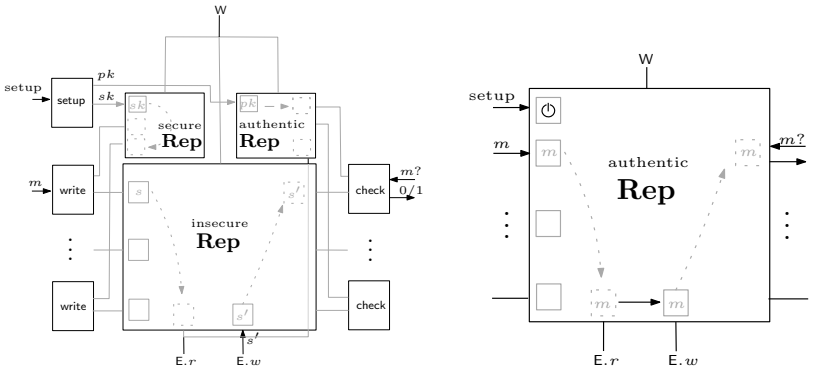


Figure 7.2: Illustration of the main construction that characterizes a digital signature scheme. The assumed resources with the protocol (left) and the constructed resource (right). The adversarial interfaces are denoted by $\mathtt{E}.w$ (write) and $\mathtt{E}.r$ (read) and the free interface is denoted by $\mathtt{W}$. The protocol is applied at the honest users' interfaces of the assumed resources.

7.1.6 The Constructive Cryptography Setting

Our construction statements are phrased again for the special case of constructive cryptography with one dishonest interface called $\mathtt{E}$, one free interface $\mathtt{W}$. See Sections 2.3.4 and 2.3.5 for the basic definitions. In contrast to the previous chapters, we consider specifications [MR16] of reactive discrete systems, meaning systems that are not fully specified. For better accessibility, we repeat here the basic concepts of specifications from [MR16] and continue the discussion from Section 2.3.6 to show how we use the concept here.

Specifications and relaxed specifications. Our specifications can be understood in the sense of game equivalence: we define an event on the inputs (and outputs) of the discrete system, and the specification states that a system must show a certain specified behavior until the condition is fulfilled, but may deviate arbitrarily afterward.

The constructive security statements can then be understood as follows. A protocol constructs from a specification $\mathcal{S}$ another specification $\mathcal{T}$ if for each system $\mathbf{S}$ that satisfies $\mathcal{S}$ there exists a system $\mathbf{T}$ that satisfies $\mathcal{T}$

such that the protocol constructs $\mathbf{T}$ from $\mathbf{S}$.

While game equivalence in general is defined based on an arbitrary MBO of the system, the MBOs considered in this paper will be of a specific and simple form: they only depend on the order in which specific inputs are given to the systems. This formalizes the guarantee that the resource behaves according to the specification if the inputs have been given in that order. A stronger condition therefore corresponds to a weaker specification, and it is easy to see that if a protocol constructs $\mathcal{T}$ from $\mathcal{S}$, and the same additional condition is specified to obtain weakened specifications $\mathcal{S}^-$ from $\mathcal{S}$ and $\mathcal{T}^-$ from $\mathcal{T}$, then the same protocol also constructs $\mathcal{T}^-$ from $\mathcal{S}^-$. (This assumes that $\mathcal{S}^-$ and $\mathcal{T}^-$ are weakened in the same way. The statement can equivalently be seen as requiring the distinguishing advantage to be small only for a subset of distinguishers.)

As the specifications in this work, as described above, can be seen as partially defined discrete systems, we use the same notation, i.e., boldface fonts. In particular, we can understand equation (2.1) as extending to such partially defined discrete systems, by changing the system to respond with a constant output to the distinguisher once the MBO has been provoked.

7.2 Message Repositories

We formalize the message repositories described above. Due to the similarity to abstract data types, we identify each exported capability by an explicit function (that can change the state of the system as a side-effect). We stick to this description format for the remainder of this chapter.

7.2.1 Description of Message Repositories

We consider general message repositories that export a certain capability, such as reading or writing a single message, at each of its interfaces. There are four types of ways in which one can access the repository to read or write its content: each interface $\mathtt{A} \in \mathcal{W}$ allows to insert one message into the repository. Interface $\mathtt{B} \in \mathcal{R}$ allows to read a message that has been written to the repository and made visible for $\mathtt{B}$. Each interface $\mathtt{C} \in \mathcal{C}$ allows to write values into the repository by specifying from which (reader) interfaces the values should be copied; no new values can be inserted at interface $\mathtt{C}$. For each copy-interface, there is a set of associated

read-interfaces from which they can copy. Each interface $\mathtt{V} \in \mathcal{V}$ allows to verify whether a certain value m is visible at the interface; this can be seen as a restricted type of read access. Finally, the free interface $\mathtt{W}$ allows to manage the visibility of messages. On a call TRANSFER$(\mathtt{A}, \mathtt{B})$, the message written at $\mathtt{A}$ becomes visible at reader interface $\mathtt{B}$. We often call the receiving interfaces *the receivers*. A precise specification of the repository appears in Figure 7.3. As indicated by the keyword **Assume**, the behavior of the repository may be undefined if this assumption is not fulfilled as discussed in Section 6.1. In contrast, "$\triangleright\ m \in \mathcal{M}$" is to be understood as a reminder or comment for the reader; the input m given to the system is necessarily in the alphabet $\mathcal{M}$ by definition of the system. (More technically, while the condition in **Assume** may be violated by an input, which may provoke an MBO, $m \in \mathcal{M}$ will always be satisfied.)

Note that one can easily generalize this basic specification to other types of read- or write-interfaces, for example to model output of partial information about a message, such as the length, but which we do not consider here and consider it as part of future work. Following the motivation of Section 6.1, for generality, we consider each described operation as associated with a separate interface.[2]

Definition 7.2.1. For finite and pairwise disjoint sets $\mathcal{W}, \mathcal{R}, \mathcal{C}, \mathcal{V}$, and a family $\{\mathcal{R}_\mathtt{C}\}_{\mathtt{C}\in\mathcal{C}}$ of sets $\mathcal{R}_\mathtt{C} \subset \mathcal{R}$ for all $\mathtt{C} \in \mathcal{C}$, we define the repository $\mathbf{Rep}^{\mathcal{C},\mathcal{W}}_{\mathcal{R},\mathcal{V},\{\mathcal{R}_\mathtt{C}\}_{\mathtt{C}\in\mathcal{C}}}$ as in Figure 7.3. For later reference, we define for $n, m, \ell, k \in \mathbb{N}$, the standard sets $\mathcal{W} = \{\mathtt{A}_i\}_{i\in[n]}$, $\mathcal{R} = \{\mathtt{B}_i\}_{i\in[\ell]}$, $\mathcal{C} = \{\mathtt{C}_i\}_{i\in[m]}$ and $\mathcal{V} = \{\mathtt{V}_i\}_{i\in[k]}$. If nothing else is specified, these standard interface names are used. We define the shorthand notation $\mathbf{Rep}^{m,n}_{\ell,k} := \mathbf{Rep}^{\mathcal{C},\mathcal{W}}_{\mathcal{R},\mathcal{V},\{\mathcal{R}_\mathtt{C}\}_{\mathtt{C}\in\mathcal{C}}}$ for these standard sets and $\mathcal{R}_\mathtt{C} = \mathcal{R}$ for all $\mathtt{C} \in \mathcal{C}$. For $\mathcal{C} = \emptyset$ we use the simplified notation $\mathbf{Rep}^{\mathcal{W}}_{\mathcal{R},\mathcal{V}}$.

Different security guarantees can be expressed using this repository by considering different allocations of read-, write-, or transfer-interfaces to different parties as discussed in the introduction. For instance, an attacker could have access to both read- and write-interfaces, to model traditional insecure communication. If the attacker only has access to read-interfaces (but not to write-interfaces beyond potentially copy-interfaces to forward

[2]Recall that it is always possible to merge several existing interfaces into one interface to model that a party or the attacker, in a certain application scenario, has the capability to write and read many messages.

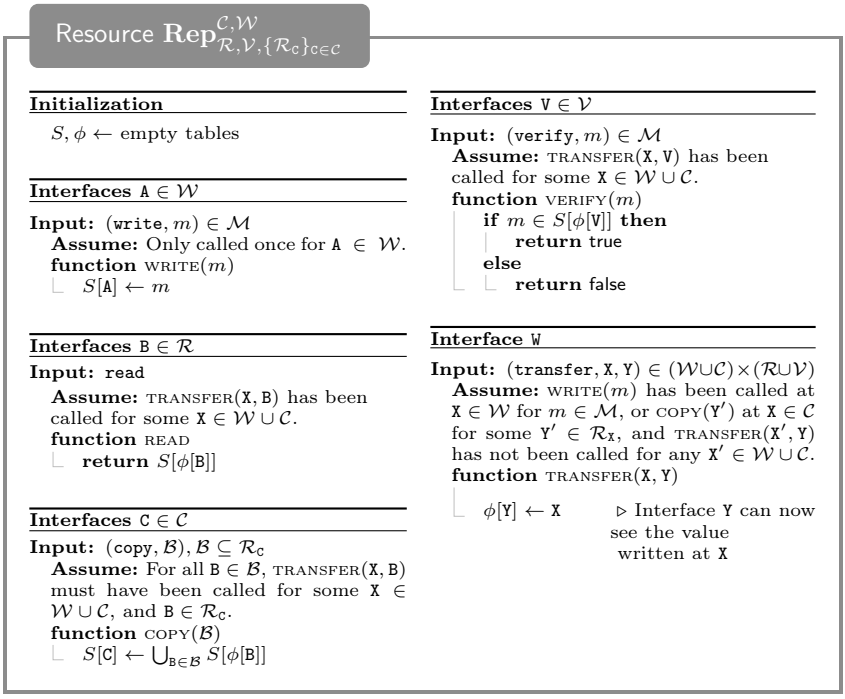


Figure 7.3: Specification of a repository resource. For ease of notation, we treat values $m \in \mathcal{M}$ and singular sets $\{m\}$ for $m \in \mathcal{M}$ interchangeably.

received messages), the repository corresponds to authenticated message transmission from a honest write-interface.

7.2.2 Modeling Security Guarantees by Access to the Repository

For security statements we need to associate each (non-free) interface to either an honest party or a possible attacker. As additional notation, we define the adversarial interfaces sets $\mathcal{E}_r := \{\mathtt{E}_1.r, \ldots, \mathtt{E}_k.r\}$ (for some $k > 0$), $\mathcal{E}_w := \{\mathtt{E}_1.w, \ldots \mathtt{E}_k.w\}$, and $\mathcal{E}_c := \{\mathtt{E}_1.c, \ldots, \mathtt{E}_k.c\}$ where the size k of this set is typically defined by the context. We can then specify repositories with different security guarantees.

- Insecure repositories allow adversarial write and read access. They can be described by $\mathbf{Rep}^{\emptyset,\mathcal{W}\cup\mathcal{E}_w}_{\mathcal{R}\cup\mathcal{E}_r,\emptyset}$, which means that all interfaces are either read- or write-interfaces.

- An authenticated repository disallows adversarial write-operations of arbitrary messages. Only (the honest) interface $\mathcal{W}$ can input content into the repository. This situation is described by the resource $\mathbf{Rep}^{\mathcal{E}_c,\mathcal{W}}_{\mathcal{R}\cup\mathcal{E}_r,\emptyset,\{\mathcal{E}_r\}_{c\in\mathcal{E}_c}}$, which indicates that the attacker may still be able to copy values from interfaces $\mathcal{E}_r$ at each interface $\mathcal{E}_c$.

- A repository without adversarial read-access, but with write access, models perfect confidentiality, and is described by $\mathbf{Rep}^{\emptyset,\mathcal{W}\cup\mathcal{E}_w}_{\mathcal{R},\emptyset}$.

While the (natural) variants described above will be the only ones used in this work, the formalism allows to flexibly define various further combinations of honest-user and adversarial capabilities.

7.3 The Constructive Perspective

7.3.1 The Basic Definitions

Our security definition for DSSs is based on the repositories introduced in Section 7.2. Intuitively, the honest parties execute a protocol to construct from an insecure repository, in which the attacker has full write access, one repository that allows the writer to authenticate a single message

(this will be used for the verification key), and one repository that allows to store a single message securely (this will be used for the signing key), an authenticated repository that can be used for multiple messages. We generally use the notation introduced in Section 7.2. We first introduce the specifications that capture authenticated repositories since they are of primary interest in this section. The first type considers repositories where the role of the receiver interfaces is to verify values in the repository:

Definition 7.3.1. Let $\mathcal{W}, \mathcal{R}, \mathcal{E}_w, \mathcal{E}_r$ denote the standard interface names. A specification $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$, in the sense of a partially defined discrete system, is an *authenticated repository for verification* if the following conditions are fulfilled. (1) It has at least the interfaces $\mathcal{I} = \mathcal{W} \cup \mathcal{R} \cup \mathcal{E}_w \cup \mathcal{E}_r$, where all inputs at $I \notin \mathcal{I}$ are ignored (i.e., the resource has the default behavior of directly returning back to the caller). (2) For all inputs at some interface $I \in \mathcal{I}$, the behavior is identical to the one specified in $\mathbf{Rep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R},\{\mathcal{E}_r\}_{\mathsf{c}\in\mathcal{E}_w}}$ for I, wherever the behavior of $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$ is defined. More formally, this means that for a given sequence of inputs, the conditional distribution of $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$, where the outputs for inputs at interfaces not in $\mathcal{I}$ are marginalized, is the same as the conditional distribution of $\mathbf{Rep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R},\{\mathcal{E}_r\}_{\mathsf{c}\in\mathcal{E}_w}}$ without those inputs.

The second definition is analogous and considers repositories where the role of the receiver interfaces is to authentically receive values:

Definition 7.3.2. Let $\mathcal{W}, \mathcal{R}, \mathcal{E}_w, \mathcal{E}_r$ denote the standard interface names. The specification $\overline{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r\cup\mathcal{R}}$, in the sense of a partially defined discrete system, is an *authenticated repository for receiving* if it has at least the interfaces $\mathcal{I} = \mathcal{W} \cup \mathcal{R} \cup \mathcal{E}_w \cup \mathcal{E}_r$, all inputs at $I \notin \mathcal{I}$ are ignored, and for all inputs at some interface $I \in \mathcal{I}$ the behavior is identical to the one specified in $\mathbf{Rep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r\cup\mathcal{R},\emptyset,\{\mathcal{E}_r\}_{\mathsf{c}\in\mathcal{E}_w}}$ for I, wherever the behavior of $\overline{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r\cup\mathcal{R}}$ is defined. We omit $\mathcal{E}_w$ in the notation if it is equal to $\emptyset$.

In the following, whenever referring to the sets $\mathcal{W}, \mathcal{R}, \mathcal{E}_w$, and $\mathcal{E}_r$, we implicitly refer to the standard names introduced in the previous section.

Assumed resources. As outlined in Section 6.1, to construct an authenticated repository, we require (beyond an insecure repository to transmit the signatures) an additional resource that allows to distribute one value authentically to all verifiers and one value securely to all signers. This

assumed communication is described by the specification $\bar{\mathbf{a}}\mathbf{Rep}^{\mathtt{S}}_{\mathcal{W}}$, which specifies resources with one writer interface $\mathtt{S}$ and no active adversarial interface. Information can only be transferred from $\mathtt{S}$ to the interfaces of $\mathcal{W}$.

To model the authenticated (but not confidential) transmission of a value, we assume another resource as specified by $\bar{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_c,\mathtt{S}}_{\mathcal{E}_r\cup\mathcal{R}}$ where information can only be transferred from $\mathtt{S}$ to the interfaces in $\mathcal{R}$, but is not limited to those as also adversarial interfaces may read this value or copy it via the interfaces in $\mathcal{E}_c$. We define the assumed system as consisting of the two above-described resources, i.e., two authenticated repositories where the first in particular models confidentiality according toSection 7.2.2, and an insecure repository $\mathbf{Rep}^{\emptyset,\mathcal{W}\cup\mathcal{E}_w}_{\mathcal{R}\cup\mathcal{E}_r,\emptyset}$. This can be succinctly summarized by the formula

$$\mathbf{R}_{n,\ell} := [\bar{\mathbf{a}}\mathbf{Rep}^{\mathtt{S}}_{\mathcal{W}}, \bar{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_c,\mathtt{S}}_{\mathcal{E}_r\cup\mathcal{R}}, \mathbf{Rep}^{\emptyset,\mathcal{W}\cup\mathcal{E}_w}_{\mathcal{R}\cup\mathcal{E}_r,\emptyset}] \,. \tag{7.1}$$

For clarity, whenever we explicitly refer to the assumed mechanism to distribute the keys, we use the shorthand notation

$$\mathbf{Dist} := [\bar{\mathbf{a}}\mathbf{Rep}^{\mathtt{S}}_{\mathcal{W}}, \bar{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_c,\mathtt{S}}_{\mathcal{E}_r\cup\mathcal{R}}.$$

Protocol converters. We assign one converter to each of the three roles: a converter write for the (honest) writer interfaces, a converter check for the (honest) reader interfaces and a setup-converter setup at interface $\mathtt{C}$. We define the vector of converters $\mathsf{DSS} := (\mathsf{setup}, \mathsf{write}, \ldots, \mathsf{write}, \mathsf{check}, \ldots, \mathsf{check})$ with n copies of write, ℓ copies of converter check and one converter setup. The set of honest interfaces in this section is defined as $\mathcal{P} := \{\mathtt{S}\} \cup \mathcal{W} \cup \mathcal{R}$. See Figure 7.4 for a formal description. Note that in the style of this chapter, we associate to each input a function that is executed upon respective input.

Goal of construction: an authenticated repository. Intuitively, the use of a DSS should allow us to construct from a repository $\mathbf{Rep}^{\mathcal{W}\cup\mathcal{E}_w}_{\mathcal{R}\cup\mathcal{E}_r}$ that allows both the honest users and the attacker to write multiple messages, and a repository that exclusively allows one honest user to write the verification key authentically, a repository in which the attacker has no write access. The reason is that writing a message that will be accepted by honest readers requires to present a valid signature relative to the

verification key, thus the attacker would be required to forge signatures. This intuition does, however, not quite hold.

Indeed, when using the insecure repository, the attacker can still copy valid signatures generated by the honest writer to which he has read access via any of his write interfaces. Since honest readers may later gain read access to those copied signatures, the attacker can indeed control *which* of the messages originating from the honest writer will be visible at those interfaces. The repository that is actually constructed is a specification $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$ as in Definition 7.3.1. The goal of a digital signature scheme can thus be understood as amplifying the capabilities of authenticated repositories as defined using the specifications above.

To give a more concrete intuition, a particular constructed resource still has an interface $\mathtt{S}$ and accepts queries $\textsc{transfer}(\mathtt{S}, \mathtt{A}_i)$ and $\textsc{transfer}(\mathtt{S}, \mathtt{B}_j)$, in addition to those provided by $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$. Providing input at these interfaces has no effect, but may influence whether further outputs of the system are still defined (because, e.g., inputs to the system may have been provided in an order such that the behavior of the DSS is not defined).

In the remainder of the section, we prove an equivalence between the validity of the described construction and the definition of existential unforgeability. As the protocol converters described above do not exactly match the algorithms in the traditional definition of a DSS, we also explain how to convert between the two representations of a signature scheme.

7.3.2 Unforgeability of Signatures implies Validity of Construction

The constructed specification $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$ has further (inactive) interfaces beyond those in $\mathcal{I} = \mathcal{W} \cup \mathcal{R} \cup \mathcal{E}_w \cup \mathcal{E}_r$, and behaves equivalently to $\mathbf{Rep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R},\{\mathcal{E}_r\}_{\mathfrak{C}\in\mathcal{E}_w}}$, as long as the assumed order of inputs is respected. The following theorem states that any existentially unforgeable digital signature scheme can be used to construct such an authentic repository from the assumed resources (see also Figure 7.2 for a depiction of this statement). Constructing a specification $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$ according to Definition 7.3.1 can be a vacuous statement: the specification can be undefined for all possible orders of inputs. The statement we prove in this section, therefore, explicitly specifies for which orders $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$ is defined. In particular, the specification is defined for all orders of inputs for which the underlying

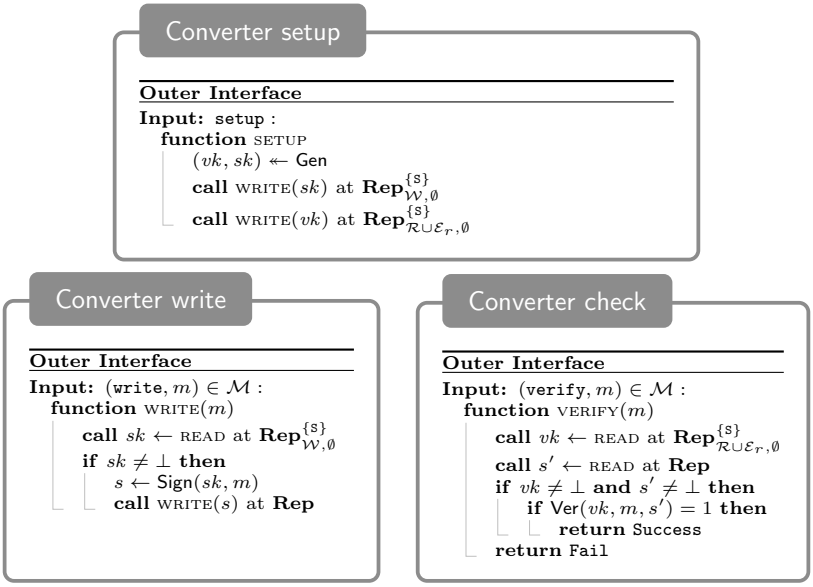


Figure 7.4: The three protocol converters derived from a signature scheme $\mathcal{DS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$.

specifications $\bar{\mathbf{a}}\mathbf{Rep}^{\mathsf{s}}_{\mathcal{W}}$ and $\bar{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_c,\mathsf{s}}_{\mathcal{E}_r\cup\mathcal{R}}$ are defined, plus the following natural conditions of a DSS: the keys are generated first and are distributed before anything is signed or verified at a writer or reader interface. As long as these conditions are satisfied, the specification defines the output of the resource. We now state the theorem.

Theorem 7.3.3. *Let $n, \ell \in \mathbb{N}$. For any given digital signature scheme $\mathcal{DS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$, let the converters* write, check, *and* setup *be defined as in Figure 7.4. Then, for the simulator* sim *defined in Figure 7.5, we design a reduction $\rho(\mathbf{D}) := \mathbf{DC}$ with a specific system $\mathbf{C}$ defined in the proof, such that for all distinguishers $\mathbf{D}$ and their associated adversaries $\mathcal{A} := \rho(\mathbf{D})$ (against the forgery game for signatures) it holds that*

$$\Delta^{\mathbf{D}}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell},\ \mathsf{sim}^{\mathrm{E}}\,\mathbf{aRep}_{n,\ell}) \leq \mathsf{Adv}^{\textit{eu-cma}}_{\mathcal{DS},\mathcal{A}}$$

and where $\mathbf{aRep}_{n,\ell}$ *is defined as long as the assumed specification is defined and the following conditions hold:*

- *Command* SETUP *is issued at the* $\mathtt{S}$*-interface before any other command;*
- *Command* TRANSFER$(\mathtt{S}, \mathtt{A}_i)$ *is issued at the* $\mathtt{W}$*-interface corresponding to the first setup repository before* WRITE *is issued at the* $\mathtt{A}_i$*-interface;*
- *Command* TRANSFER$(\mathtt{S}, \mathtt{B}_i)$ *is issued at the* $\mathtt{W}$*-interface corresponding to the second setup repository before* READ *is issued at the* $\mathtt{B}_i$*-interface.*
- *There are no* TRANSFER$(\mathtt{X}, \mathtt{Y})$ *queries with* $\mathtt{X} \in \mathcal{E}_w$ *and* $\mathtt{Y} \in \mathcal{E}_r$*, that is, we exclude communication from the adversarial writer to adversarial reader-interfaces.*

In other words, the signature scheme constructs the specification $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$ from the assumed specification $\mathbf{R}_{n,\ell}$.

Proof. Consider the simulator given in Figure 7.5. We introduce the shorthand notation $\mathbf{S}_{n,\ell} := \mathsf{sim}^{\mathtt{E}}\,\mathbf{aRep}_{n,\ell}$. The simulator sim accesses all the capabilities provided at the adversarial interfaces (by merging them into the single interface $\mathtt{E}$) and provides at its outer interface the simulated capabilities (accessible at the appropriate interfaces) of the real system. We now argue that all queries of a distinguisher $\mathbf{D}$ are answered consistently by the systems $\mathbf{S}_{n,\ell}$ and $\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}$, we observe that in random experiments $\mathbf{D}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})$ and $\mathbf{D}(\mathbf{S}_{n,\ell})$, the behavior is identical, unless a query WRITE(s') is made at interface $\mathtt{E}$ for some signature s' that satisfies $\mathsf{Ver}(vk, m', s') = 1$ for some message m' that is subsequently part of a call VERIFY(m') at the interface where s' was transferred to. Let us denote this event in the real-world random experiment by BAD. If we prove that the systems behave equivalently until event BAD occurs, then we can bound the distinguishing advantage of $\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}$ and $\mathbf{S}_{n,\ell}$ by bounding the probability of event BAD. As discussed above, this probability can be bounded by the success probability of an (efficient) adversary $\mathcal{A}$ in breaking the security of the DSS.

We now elaborate on the other queries being handled consistently by the two systems and argue for each invoked function individually.

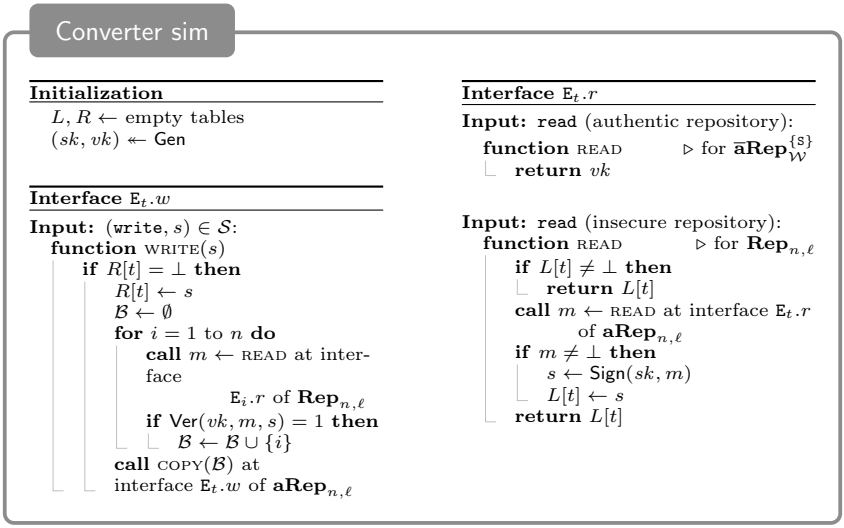


Figure 7.5: Simulator for the proof of Theorem 7.3.3.

S.SETUP: This query has no effect in the ideal system, but in the real system it makes the key sk available in $\overline{\mathbf{a}}\mathbf{Rep}_{\mathcal{W}}^{\{\mathtt{S}\}}$ and the key vk available in $\overline{\mathbf{a}}\mathbf{Rep}_{\mathcal{E}_r \cup \mathcal{R}}^{\mathcal{E}_c, \{\mathtt{S}\}}$. No output to $\mathbf{D}$.

W.TRANSFER$(\mathtt{S}, \mathtt{A})$ for $\mathtt{A} \in \mathcal{W}$: Requires that S.SETUP has been called. No effect in the ideal system, but in the real system it makes the key sk accessible to writer $\mathtt{A}$. No output to $\mathbf{D}$.

W.TRANSFER$(\mathtt{S}, \mathtt{X})$ for $\mathtt{X} \in \mathcal{E}_r$ at $\overline{\mathbf{a}}\mathbf{Rep}_{\mathcal{E}_r \cup \mathcal{R}}^{\mathcal{E}_c, \{\mathtt{S}\}}$: Requires that S.SETUP has been called. No effect in the ideal system, but in the real system it makes the key vk accessible to adversarial reader $\mathtt{X}$. No immediate output to $\mathbf{D}$.

Y.COPY$(\mathtt{X})$ for $\mathtt{X} \in \mathcal{E}_r$ and $\mathtt{Y} \in \mathcal{E}_c$ at $\overline{\mathbf{a}}\mathbf{Rep}_{\mathcal{E}_r \cup \mathcal{R}}^{\mathcal{E}_c, \{\mathtt{S}\}}$: Requires that before query W.TRANSFER$(\mathtt{S}, \mathtt{X})$ has been made. No effect in the ideal system, but in the real system it copies the key vk to interface $\mathtt{Y}$. No immediate output to $\mathbf{D}$.

W.**TRANSFER**(X, B) **for** B $\in \mathcal{R}$ **at** $\bar{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_c,\{S\}}_{\mathcal{E}_r\cup\mathcal{R}}$**:** Requires that either S.SETUP resp. X.COPY has been called. No effect in the ideal system, but in the real system it makes the key vk accessible to reader B. No output to **D**.

A.**WRITE**(m) **for** A $\in \mathcal{W}$**:** Requires that W.TRANSFER(S, A) has been called. Enters m into the resource in the ideal system, obtains vk, computes the signature, and enters it into $\mathbf{Rep}^{\mathcal{W}\cup\mathcal{E}_w}_{\mathcal{R}\cup\mathcal{E}_r}$ in the real system. No immediate output to **D**.

X.**READ for** X $\in \mathcal{E}_r$ **at** $\bar{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_c,\{S\}}_{\mathcal{E}_r\cup\mathcal{R}}$**:** Requires that W.TRANSFER(S, X) has been called. In the real system, returns the verification key vk. In the ideal system, sim returns the simulated verification key. This has the same distribution.

W.**TRANSFER**(A, X) **for** X $\in \mathcal{R} \cup \mathcal{E}_r$**:** Requires that A.WRITE(m) has been called. In the real system, makes the generated signature available to X. In the ideal system, makes m available to X. No immediate output to **D**.

X.**READ for** X $\in \mathcal{E}_r$ **at** $\mathbf{Rep}^{\mathcal{W}\cup\mathcal{E}_w}_{\mathcal{R}\cup\mathcal{E}_r}$**:** Requires that W.TRANSFER(A, X) has been queried for A $\in \mathcal{W}$. (Adversarial writers in $\mathcal{E}_w$ are explicitly excluded.) In the real system, outputs the signature that has been made available in the repository. In the ideal system, the simulator sim checks whether the query has been made before, answers consistently in that case, and otherwise generates a new signature using the internal key. This computation is done exactly in the same way, and therefore the returned signature also has the same distribution.

X.**WRITE**(s) **for** X $\in \mathcal{E}_w$**:** In the real system, this enters the value s into the repository $\mathbf{Rep}^{\mathcal{W}\cup\mathcal{E}_w}_{\mathcal{R}\cup\mathcal{E}_r}$ and has no immediate output. In the ideal system, the simulator sim processes the message, and checks for which messages that it has already received, the signature verifies. In case no message can be verified, the simulator inserts $\perp$ at its copier interface. In the other case, for each message, received at reader-subinterface i that verifies successfully with the given signature string, the simulator calls COPY(i) at X to insert this message into the copier buffer (since all these messages can be verified w.r.t. the signature string s.)

$\mathtt{W}.\textsc{transfer}(\mathtt{X},\mathtt{B})$**:** This is only valid if $\mathtt{X}.\textsc{write}(s)$ was called before, has the same effect in both cases.

We introduce the reduction system $\mathbf{C}$ that emulates the real world view towards any distinguisher $\mathbf{D}$ by accessing the oracles of $\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}$, i.e., such that $\mathbf{C}^{\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}}$ and $\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}$ have the same input-output behavior. During that emulation, $\mathbf{C}$ tries to extract a forgery from the interaction with the distinguisher. The main challenge for the reduction is to make sure that it does only query the signing oracle on messages that definitely cannot be forgery candidates. Formally, the reduction system $\mathbf{C}$ emulates one setup interface, n writer-interfaces $\mathtt{A}_t$, ℓ receiver-interfaces $\mathtt{B}_t$, and $n+\ell$ adversarial read and write interfaces $\mathtt{E}_t.r/w$ and one free interface $\mathtt{W}$ towards $\mathbf{D}$. $\mathbf{C}$ first initializes two empty tables R and L. Then, $\mathbf{C}$ receives the verification key vk from $\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}$ and stores it internally. Furthermore, it answers all queries by $\mathbf{D}$ accordingly to the description of the real system, with the only difference that signatures are computed through the game. When $\mathbf{C}$ detects a signature forgery, it outputs this forgery to the game.

For any distinguisher $\mathbf{D}$, we define the the adversary $\mathcal{A} := \mathbf{DC}$ against the game $\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}$ and conclude that

$$\begin{aligned}\Delta^{\mathbf{D}}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}, \mathbf{S}_{n,\ell}) &\leq \Pr^{\mathbf{D}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})}[BAD] = \Pr^{\mathbf{DC}^{\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}}}[BAD] \\ &= \Pr\left[\mathbf{DC}^{\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}} \text{ sets } \mathsf{win}\right] = \mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathcal{DS},\mathcal{A}}.\end{aligned}$$

□

7.3.3 Chaining Multiple Construction Steps

The construction proved in Theorem 7.3.3 assumes (amongst others) an authenticated repository $\bar{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_c,\mathsf{S}}_{\mathcal{E}_r\cup\mathcal{R}}$ and constructs an authenticated repository $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$. A natural question is in which sense multiple such construction steps can be chained, corresponding to signing the verification key of one instance with a different instance of the scheme. For this to work out, we have to "upgrade" the resource $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$ to a resource $\bar{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r\cup\mathcal{R}}$ as needed by Theorem 7.3.3, where we can then use any interface $\mathsf{X} \in \mathcal{W}$ as the interface $\mathtt{S}$ to transmit the secret key. Of

course, we additionally require resources $\overline{\mathbf{a}}\mathbf{Rep}^{\{\mathsf{X}\}}_{\mathcal{W}'}$ for distributing the secret keys and $\mathbf{Rep}^{\emptyset,\mathcal{W}'\cup\mathcal{E}'_w}_{\mathcal{R}\cup\mathcal{E}_r,\emptyset}$ for transmitting the signatures.

The chaining is then achieved by the protocol that consists of converters send and $\mathsf{receive}$, sends the messages over an (additional) insecure repository $\mathbf{Rep}^{\emptyset,\mathcal{W}\cup\mathcal{E}_w}_{\mathcal{R}\cup\mathcal{E}_r,\emptyset}$ and authenticates them via $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$. Protocol converter send simply inputs the same message to both resources, whereas $\mathsf{receive}$ verifies the messages obtained through the insecure repository at the authenticated repository. This protocol perfectly constructs an authenticated repository with delivery from the two assumed resources.

Theorem 7.3.4. *Let $n, \ell \in \mathbb{N}$, and consider a protocol SND with converters send for all interfaces in $\mathcal{W}$ and converters $\mathsf{receive}$ for all interfaces in $\mathcal{R}$, defined as described above. Then, for the simulator sim described below we have for any distinguisher $\mathbf{D}$,*

$$\Delta^{\mathbf{D}}(\mathsf{SND}_{\mathcal{P}}[\mathbf{Rep}^{\emptyset,\mathcal{W}\cup\mathcal{E}_w}_{\mathcal{R}\cup\mathcal{E}_r,\emptyset}, \mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}], \mathsf{sim}^{\mathsf{E}}\overline{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r\cup\mathcal{R}}),$$

wherever both resources are defined. The constructed resource $\overline{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r\cup\mathcal{R}}$ accepts TRANSFER *commands at sub-interfaces corresponding to both assumed resources, and requires, for a given message to be transferred, both those commands to be issued.*

Proof Sketch. The simulator sim responds to READ queries at the $\mathcal{E}_r$-interfaces corresponding to $\mathbf{Rep}^{\emptyset,\mathcal{W}\cup\mathcal{E}_w}_{\mathcal{R}\cup\mathcal{E}_r,\emptyset}$ or $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$ by obtaining the transmitted messages from $\overline{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r\cup\mathcal{R}}$. Once COPY has been called at an $\mathcal{E}_w$-interface at $\mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$ and the corresponding message has been input at the same $\mathcal{E}_w$-interface of $\mathbf{Rep}^{\emptyset,\mathcal{W}\cup\mathcal{E}_w}_{\mathcal{R}\cup\mathcal{E}_r,\emptyset}$, sim issues the same COPY command at $\overline{\mathbf{a}}\mathbf{Rep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r\cup\mathcal{R}}$. □

Together with Theorem 7.3.3, this means that sending a message along with a signature constructs an authenticated repository from which the authenticated messages can be read. Several such constructions can then be chained in the expected way.

7.3.4 Validity of Construction implies Unforgeability of Signatures

In this section, we show that any converters achieving the construction of **aRep** from **Rep** and **Dist** contain a digital signature scheme that is

existentially unforgeable under chosen-message attacks. More precisely:

Theorem 7.3.5. *Let $n, \ell \in \mathbb{N}$. Consider arbitrary converters* setup, write, *and* check *and define* $\mathsf{DSS} := (\mathsf{setup}, \mathsf{write}, \ldots, \mathsf{write}, \mathsf{check}, \ldots, \mathsf{check})$ *(for the honest interfaces) with n copies of* write, *ℓ copies of converter* check *and one converter* setup. *We derive a digital signature scheme $\mathcal{DS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ below in Figure 7.6 with the following property: given any adversary against the signature scheme that asks at most n signing queries and makes at most ℓ forgery queries to the oracles of $\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}$, we construct (efficient) distinguishers $\mathbf{D}_i$, $i = 1 \ldots 5$, such that for the systems $\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}$ and $\mathsf{sim}^{\mathrm{E}}\mathbf{aRep}_{n,\ell}$, with $\mathbf{aRep}_{n,\ell} = \mathbf{aRep}^{\mathcal{E}_w,\mathcal{W}}_{\mathcal{E}_r,\mathcal{R}}$, for all simulators* sim,

$$\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathcal{DS},\mathcal{A}} \leq \sum_{i=1}^{5} \Delta^{\mathbf{D}_i}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell},\ \mathsf{sim}^{\mathrm{E}}\,\mathbf{aRep}_{n,\ell}),$$

and where $\mathbf{aRep}_{n,\ell}$ is defined as long as the assumed specification is defined and under the same additional conditions as in Theorem 7.3.3.

Obtaining the signature scheme from the converters. The key generation, signing, and verification functions are derived from the converters setup, write, and check that construct **aRep** from [**Dist**, **Rep**] as follows: The key generation Gen consists of evaluating the function setup.SETUP, the two values written to the resource **Dist** are considered as the corresponding key pair. The secret key is the value that is written to the first sub-system of **Dist**. The signing algorithm $\mathsf{Sign}(sk, m)$ consists of evaluating the function write.WRITE(m). The signature for message m is defined as the value that is written to the repository. Any request to obtain a value from resource **Dist** is answered by providing the signing key sk. The verification algorithm $\mathsf{Ver}(vk, m, s)$ consists of evaluating the function check.VERIFY(m) and the candidate signature s is provided as the actual value in the repository and the verification key vk is given as the value in **Dist**. The formal description of the algorithms appear in Figure 7.6.

Proof. The theorem directly follows from the following two observations: Lemma 7.3.6 states that if the output of the key generation algorithm Gen as defined in Section 7.3.4 is not $(\perp, \perp)$, then any adversary against

Key Generation

function Gen
  sk $\leftarrow \perp$; vk $\leftarrow \perp$
  evaluate setup.SETUP:
    - *Reaction on call* WRITE(val_s) to **Dist** (first subsystem):
      if sk $= \perp$ **then**
        sk $\leftarrow val_s$
    - *Reaction on call* WRITE(val_r) to **Dist** (second subsystem):
      if vk $= \perp$ **then**
        vk $\leftarrow val_r$
  if sk $\neq \perp$ **and** vk $\neq \perp$ **then**
    return (sk, vk)
  else
    return $(\perp, \perp)$

Signature Generation

function Sign(sk, m)
  if $sk = \perp$ **then**
    return m
  $s \leftarrow \perp$
  evaluate write.WRITE(m):
    - *Reaction on call* READ to **Dist**:
      Return sk to write.WRITE
    - *Reaction on call* WRITE($\tilde{m}$) to **Rep**:
      if $s = \perp$ **then**
        $s \leftarrow \tilde{m}$
  if $s \neq \perp$ **then**
    return s
  else
    return m

Signature Verification

function Ver(vk, m, s)
  if $vk = \perp$ **then**
    return $s \stackrel{?}{=} m$
  evaluate RES $\leftarrow$ check.VERIFY(m):
    - *Reaction on call* READ to **Dist**:
      Return vk to check.VERIFY
    - *Reaction on call* READ to **Rep**:
      Return s to check.VERIFY
  if RES $=$ Success **then**
    return 1
  else
    return 0

Figure 7.6: Signature scheme (Gen, Sign, Ver) extracted from converters setup, write, and check.

the derived signature scheme can be transformed into a distinguisher for the real and ideal systems (for any simulator).

Lemma 7.3.6. *Let $n, \ell \in \mathbb{N}$, let* DSS *be as defined in Theorem 7.3.5 for arbitrary converters* setup, write, *and* check, *and let the digital signature scheme $\mathcal{DS} =$ (Gen, Sign, Ver) be defined as in Section 7.3.4. We present an (efficient) reduction that transforms any adversary $\mathcal{A}$ for $\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}$, that asks at most n signing queries and ℓ forgery queries to the oracles of the forgery game, into a distinguisher $\mathbf{D}(\mathcal{A})$, such that for all simulators* sim,

$$\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathcal{DS},\mathcal{A}} \leq \Pr[(\perp,\perp) \leftarrow \mathsf{Gen}] + \Delta^{\mathbf{D}(\mathcal{A})}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell},\ \mathsf{sim}^{\mathrm{E}}\,\mathbf{aRep}_{n,\ell}).$$

Proof. We define the shorthand notation $\mathbf{S}_{n,\ell} := \mathsf{sim}^{\mathrm{E}}\,\mathbf{aRep}_{n,\ell}$. We define a reduction system $\mathbf{C}$ that is given access to the interfaces of either system

$\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}$ or $\mathbf{S}_{n,\ell}$, and provides one additional outside interface. At that outside interface, $\mathbf{C}$ simulates the signing and forgery oracles of game $\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}$. First, the system $\mathbf{C}$ initializes an internal variable WON to 0. Then, it activates all interfaces of its connected system and queries SETUP at interface $\mathtt{C}$. Subsequently, $\mathbf{C}$ calls TRANSFER$(\mathtt{C}, \mathtt{E}_1.r)$, TRANSFER$(\mathtt{C}, \mathtt{B}_i)$, and TRANSFER$(\mathtt{C}, \mathtt{A}_j)$ for $i = 1 \ldots \ell$, $j = 1 \ldots n$. Then, $\mathbf{C}$ queries READ to (the possibly simulated) resource **Dist** at interface $\mathtt{E}_1.r$ and outputs at the outside interface whatever is output by $\mathtt{E}_1.r$.

It further answers the following queries by an adversary $\mathcal{A}$:

On the ith signing query m: Upon this query, $\mathbf{C}$ queries WRITE(m) at interface $\mathtt{A}_i$ and subsequently TRANSFER$(\mathtt{A}_i, \mathtt{E}_i.r)$. Then, retrieve the value s by querying READ at interface $\mathtt{E}_i.r$. Finally, $\mathbf{C}$ outputs the pair (m, s) at its outer interface. If no value is output at interface $\mathtt{E}_i.r$, then $\mathbf{C}$ outputs (m, m).

On the ith forgery query (m, s): On input a possible forgery, $\mathbf{C}$ queries WRITE(s) at interface $\mathtt{E}_i.w$ of its connected system. Then, $\mathbf{C}$ queries TRANSFER$(\mathtt{E}_i.w, \mathtt{B}_i)$ to give $\mathtt{B}_i$ access to the signature string s. Next, $\mathbf{C}$ queries VERIFY(m) at interface $\mathtt{B}_i$ to receive either the the indication `Success` or `Fail`. If the message m is successfully verified and has not been queried to SIGN before, then WON is set to 1. In any case WON is output at the outside interface.

We first note that in the key generation process of $\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}$ for $\mathcal{DS}$ defined above, the function setup.SETUP defines the signing and verification key. It is this function that defines the values that converter setup writes to the distribution resource in the real world upon SETUP at interfaces $\mathtt{C}$. So the probability distribution of the pair (vk, sk) output by Gen is identical to the distribution of the values written to **Dist** in system $\mathbf{C}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})$.

In the random experiment of any adversary that asks at most n queries to its signature oracle and ℓ queries to its forgery oracle, the input-output behavior of $\mathbf{C}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})$ and $\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}$ are identical given that $vk \neq \perp$ and $sk \neq \perp$ during the key generation process. This follows from the definition of the algorithms of $\mathcal{DS}$: The signing algorithm Sign and the verification algorithm Ver execute the same converter functions as are executed in the system $\mathbf{C}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})$ on an input by the adversary. And in both cases, if no signature is generated for some message m it is set to

the default value m. This is sufficient to build a distinguisher based on an adversary $\mathcal{A}$.

Claim 1. *Let $n, \ell \in \mathbb{N}$. From any game winner $\mathcal{A}$ for $\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}$, that asks at most n signing queries and ℓ forgery queries, we construct a distinguisher $\mathbf{D}$ such that for any simulator sim,*

$$\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathcal{DS},\mathcal{A}} \leq \Delta^{\mathbf{D}}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell},\ \mathsf{sim}^{\mathtt{E}}\,\mathbf{aRep}) + \Pr[(\bot,\bot) \leftarrow \mathsf{Gen}].$$

Proof of claim. Consider the random experiment in which an adversary $\mathcal{A}$ interacts with system $\mathbf{C}(\mathbf{T})$, where $\mathbf{T} \in \{\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}, \mathbf{S}_{n,\ell}\}$. Let further be F the binary random variable that takes on the value 1 if at least one component of the key-pair is undefined at the point of answering the first oracle query (and $F = 0$ otherwise).

The actual distinguisher $\mathbf{D}$ connected to a system $\mathbf{T} \in \{\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}, \mathbf{S}_{n,\ell}\}$ works as follows: it lets $\mathcal{A}$ interact with system $\mathbf{C}(\mathbf{T})$ and, after $\mathcal{A}$ has finished, outputs the value of WON as its decision bit[3]

We can therefore conclude that

$$\begin{aligned}
&\Pr[\mathbf{D}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}) = 1] = \Pr\Big[\mathcal{A}^{\mathbf{C}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})} \text{ sets WON}\Big] \\
&= \Pr^{\mathcal{A}^{\mathbf{C}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})}}[F = 0] \cdot \Pr\Big[\mathcal{A}^{\mathbf{C}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})} \text{ sets WON} \,|\, F = 0\Big] \\
&+ \Pr^{\mathcal{A}^{\mathbf{C}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})}}[F = 1] \cdot \Pr\Big[\mathcal{A}^{\mathbf{C}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})} \text{ sets WON} \,|\, F = 1\Big] \\
&\leq \Pr^{\mathcal{A}^{\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}}}[F = 0] \cdot \Pr\Big[\mathcal{A}^{\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}} \text{ sets } \mathsf{win} \,|\, F = 0\Big] \\
&\quad + \Pr^{\mathcal{A}'^{\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}}}[F = 1] \cdot \underbrace{\Pr\Big[\mathcal{A}'^{\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}} \text{ sets } \mathsf{win} \,|\, F = 1\Big]}_{=1} \\
&\leq \Pr\Big[\mathcal{A}^{\mathsf{EU\text{-}CMA}^{\mathrm{sig}}_{\mathcal{DS}}} \text{ sets } \mathsf{win}\Big] + \Pr[(\bot,\bot) \leftarrow \mathsf{Gen}], \qquad (7.2)
\end{aligned}$$

where $\mathcal{A}'$ is the adversary that wins the game with probability 1 by a single query (m, m) to oracle `Forge` in case $F = 1$. Note that by definition, the scheme does not provide any security whenever this condition occurs. On the other hand, $\Pr\big[\mathbf{D}(\mathsf{sim}^{\mathtt{E}}\mathbf{aRep}_{n,\ell}) = 1\big] = 0$, since no new message can be written at any interface $\mathtt{E}_i.w$ of $\mathbf{aRep}_{n,\ell}$. This yields the claim. ◊

[3]This means that the output bit 1 indicates that the connected system is the real system.

This concludes the proof. □

The second lemma states that for the key generation algorithm Gen defined in Section 7.3.4, the probability that $(\perp, \perp)$ is returned is a lower bound for the advantage in distinguishing the real and ideal systems.

Lemma 7.3.7. *Let $n, \ell \in \mathbb{N}$, let $\mathbf{R}_{n,\ell}$ be as defined above for converters* setup, write, *and* check *and let the digital signature scheme $\mathcal{DS} =$ (Gen, Sign, Ver) be defined as in Section 7.3.4. We construct (efficient) distinguishers $\mathbf{D}_i$, $i = 1 \dots 5$, such that for all simulators* sim*,*

$$\Pr[(\perp, \perp) \leftarrow \mathsf{Gen}] \leq \sum_{i=1}^{4} \Delta^{\mathbf{D}_i}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}, \mathsf{sim}^{\mathtt{E}}\mathbf{aRep}_{n,\ell})$$

In particular, if there exists a simulator sim *such that $\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}$ and $\mathsf{sim}^{\mathtt{E}}\mathbf{aRep}_{n,\ell}$ are indistinguishable, then the output of the key generation algorithm of Figure 7.6 is defined with overwhelming probability.*

Proof. Let us consider an execution of algorithm Gen and let us define the events $\mathcal{E}_1 := sk = \perp$ and $\mathcal{E}_2 := vk = \perp$ and let $\mathcal{E} := \mathcal{E}_1 \cup \mathcal{E}_2$. By definition of algorithm Gen, we immediately observe that $\Pr[\mathcal{E}_i]$ is equal to the probability that, in the real system $\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}$ upon calling SETUP at $\mathtt{C}$, converter setup does not define the respective values by an appropriate write-query to **Dist**. We now show that the occurrence of either event leads to lower bounds on the security condition.

To achieve this, let us consider the following real-world random experiment (with system $\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}$). For further reference, we denote the experiment by *Exp*. First, we call SETUP at interface $\mathtt{C}$. Subsequently, we call TRANSFER$(\mathtt{C}, \mathtt{E}_1.r)$, TRANSFER$(\mathtt{C}, \mathtt{B}_i)$, and TRANSFER$(\mathtt{C}, \mathtt{A}_j)$ for $i = 1 \dots \ell$, $j = 1 \dots n$ to distribute the setup values (,i.e., keys). Then, we choose a uniformly random message $\overline{m}$ and input it at interface $\mathtt{A}_1$. Let us denote by S the random variable that takes on the value of the output of write on this write-query. Afterward, we call TRANSFER$(\mathtt{A}_1, \mathtt{B}_1)$.[4] Finally, we call VERIFY$(\overline{m})$ at interface $\mathtt{B}_1$. Let R be the random variable that takes on the value output by check. We decompose the probability of

[4]Note that this input bypasses any adversarial influence, as the output by the honest writer is directly given to the honest reader.

event $\mathcal{E}$ in *Exp* as follows:

$$\Pr^{Exp}[\mathcal{E}] = \underbrace{\Pr^{Exp}[\mathcal{E}] \cdot \Pr^{Exp}[S = \bot \mid \mathcal{E}]}_{\alpha} \tag{7.3}$$

$$+ \underbrace{\Pr^{Exp}[\mathcal{E}] \cdot \Pr^{Exp}[S \neq \bot \mid \mathcal{E}] \cdot \Pr^{Exp}[R = \texttt{Fail} \mid \mathcal{E}, S \neq \bot]}_{\beta} \tag{7.4}$$

$$+ \sum_{i=1}^{2} \underbrace{\Pr^{Exp}[\mathcal{E}_i] \cdot \Pr^{Exp}[S \neq \bot \mid \mathcal{E}_i] \cdot \Pr^{Exp}[R = \texttt{Success} \mid \mathcal{E}_i, S \neq \bot]}_{\gamma_i}. \tag{7.5}$$

Let $\mathbf{D}_1$ be the distinguisher that implements the strategy of *Exp* with the following exception: instead of calling VERIFY($\overline{m}$) in the last step, $\mathbf{D}_1$ flips a uniform bit b and only if $b = 0$ it calls VERIFY($\overline{m}$); if $b = 1$ $\mathbf{D}_1$ calls VERIFY(m') for with m' chosen uniformly at random from $\mathcal{M} \setminus \{\overline{m}\}$. $\mathbf{D}_1$ outputs 1 as its decision bit if either $R = \texttt{Success}$ and $b = 1$ or if $R = \texttt{Fail}$ and $b = 0$. In any other case, $\mathbf{D}_1$ outputs 0.

We see that the random variable S in the experiment between $\mathbf{D}_1$ and $\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}$ is identically distributed as in experiment *Exp*. Hence, in the real world, with probability α, converter check attached at interface $\mathtt{B}_1$ does not learn any information about $\overline{m}$, because $\overline{m}$ was chosen after the SETUP query and no information is transmitted within the repository since $S = \bot$. In this case, the probability that $\mathbf{D}_1$ outputs 1 is $\frac{1}{2}$. We observe that in the ideal world, i.e., in $\mathbf{D}_1(\mathsf{sim}^{\mathtt{E}}\mathbf{aRep}_{n,\ell})$, the output $R = \texttt{Success}$ is observed whenever $b = 0$ and the output $R = \texttt{Fail}$ is observed whenever $b = 1$. Hence,

$$\Delta^{\mathbf{D}_1}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell},\ \mathsf{sim}^{\mathtt{E}}\,\mathbf{aRep}_{n,\ell}) \geq \frac{\alpha}{2}.$$

Let $\mathbf{D}_2$ be the distinguisher that implements the strategy in experiment *Exp* and which outputs 1 if and only if $R = \texttt{Fail}$. We directly see that in the experiment between $\mathbf{D}_2$ and $\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell}$, converter check outputs $\texttt{Fail}$ with probability β, whereas in the random experiment $\mathbf{D}_1(\mathsf{sim}^{\mathtt{E}}\mathbf{aRep}_{n,\ell})$ with any simulator, the probability of an output $\texttt{Fail}$ at interface $\mathtt{B}_1$ is 0 by definition:

$$\Delta^{\mathbf{D}_1}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell},\ \mathsf{sim}^{\mathtt{E}}\,\mathbf{aRep}_{n,\ell}) \geq \beta.$$

We now show that the third term constitutes a security issue in both cases:

Event $\mathcal{E}_1$: We define a third distinguisher $\mathbf{D}_3$ as follows: it first chooses a uniformly random message $\overline{m}$, activates all interfaces, and then calls WRITE($\overline{m}$) at interface $\mathtt{A}_1$, subsequently retrieves the value of S by calling TRANSFER($\mathtt{A}_1, \mathtt{E}_1.r$) and after that calls READ at interface $\mathtt{E}_1.r$. Only then, $\mathbf{D}_3$ calls SETUP and distributes the setup values as in *Exp*. Then, $\mathbf{D}_3$ calls WRITE(S) at interface $\mathtt{E}_1.w$ and calls TRANSFER($\mathtt{E}_1.w, \mathtt{B}_1$) to give the receiver interface access to the value S.

Finally, $\mathbf{D}_3$ queries VERIFY($\overline{m}$) at interface $\mathtt{B}_1.r$. $\mathbf{D}_2$ outputs 1 if the output is `Success` and outputs 0 otherwise. We have in particular

$$\Pr^{\mathbf{D}_3(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})}[\mathcal{E}_1 \mid S \neq \perp] = \Pr^{Exp}[\mathcal{E}_1]$$
$$\Pr^{\mathbf{D}_3(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})}[S \neq \perp] = \Pr^{Exp}[S \neq \perp \mid \mathcal{E}_1]\,,$$

since in both experiments, converter write is invoked when the value val_s stored in **Dist** for converter write is $\perp$. Note that this in particular means, this behavior has to be in the specification, since the behavior of *Exp* obeys all the conditions. We conclude that the probability of an output other than `Fail` at interface $\mathtt{B}_1$ in the real world is exactly γ_1. On the other hand, in the ideal world, i.e., in $\mathbf{D}_2(\mathsf{sim}^{\mathtt{E}}\mathbf{aRep}_{n,\ell})$, there cannot be any output other than `Fail` for this distinguishing strategy. We get

$$\Delta^{\mathbf{D}_3}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell},\ \mathsf{sim}^{\mathtt{E}}\,\mathbf{aRep}_{n,\ell}) \geq \gamma_1.$$

Event $\mathcal{E}_2$: We define a fourth distinguisher $\mathbf{D}_4$ as follows: instead of querying SETUP, $\mathbf{D}_4$ internally runs setup.SETUP to simulate the public and the private values. Let the private value be denoted as sk. Then, $\mathbf{D}_4$ chooses a message $\overline{m}$ uniformly at random and simulates the output S of converter write by evaluating write.WRITE($\overline{m}$) using sk as the emulated value stored in **Dist**. $\mathbf{D}_4$ then activates all interfaces of its connected system and queries WRITE(S) at interface $\mathtt{E}_1.w$ and TRANSFER($\mathtt{E}_1.w, \mathtt{B}_1$). Finally, $\mathbf{D}_4$ queries VERIFY($\overline{m}$) at interface $\mathtt{B}_1$ and outputs 1 as its decision bit if and only if the output R is `Success`. We observe that in particular,

$$\Pr^{\mathbf{D}_4(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})}[\mathcal{E}_2] = \Pr^{Exp}[\mathcal{E}_2]$$

and

$$\begin{aligned}&\Pr^{\mathbf{D}_4(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell})}[R = \texttt{Success} \mid \mathcal{E}_2, S \neq \perp]\\ &= \Pr^{Exp}[R = \texttt{Success} \mid \mathcal{E}_2, S \neq \perp]\,.\end{aligned}$$

The first equation follows from the fact that $\mathbf{D}_4$ internally imitates the setup-process and hence the probability that the (public) value is equal to $\perp$ is the same as in *Exp*. The second equality follows since the value that converter check retrieves from **Dist** is $\perp$ in both systems and hence the views are identical (and again this shows that the behavior for this resource has to be in the specification). We conclude that the probability of an output Success at interface $\mathtt{B}_1.r$ in the real world is exactly γ_2. On the other hand, in the ideal world, i.e., in $\mathbf{D}_4(\mathsf{sim}^{\mathsf{E}}\mathbf{aRep})$, there cannot be any output other than Fail, since the message $\overline{m}$ has never been written to the repository. We get

$$\Delta^{\mathbf{D}_4}(\mathsf{DSS}_{\mathcal{P}}\mathbf{R}_{n,\ell},\ \mathsf{sim}^{\mathsf{E}}\,\mathbf{aRep}_{n,\ell}) \geq \gamma_2.$$

This concludes the proof of the second lemma. □

The above two lemmata establish the theorem statement. □

7.3.5 On the Transferability of Verification Rights

Universal verification is arguably an important property of signatures. Anybody possessing the public key and a valid signature string s for some message m can verify the signature. This implies furthermore that signatures are naturally transferable, which is essential for their key role in public-key infrastructures or signing electronic documents. In this section, we demonstrate that our definition directly implies transferability by constructing a message repository in which information can be forwarded among readers. The high-level idea is to apply a converter to the free interface that instead copies the desired message from the sender buffer, where it was input originally, to the targeted reader buffer.

The role of the free interface. Recall that the role of the free interface in the repository resources is to transfer the contents from certain write-buffers to certain read-buffers. The transferability of signatures then simply means that values can also be transferred from read-buffers to

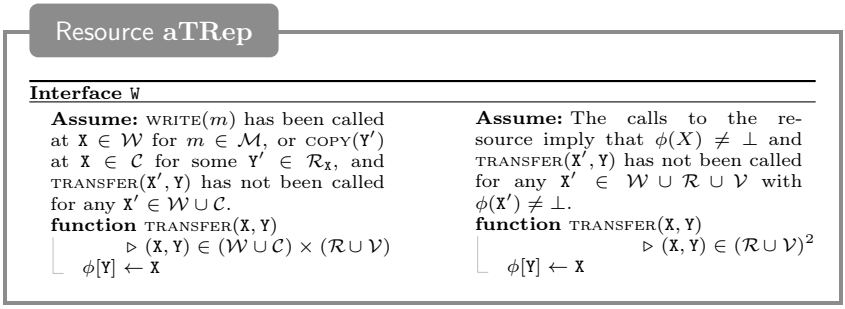


Figure 7.7: Specification of a repository resource with transferable rights. Only the modifications with respect to Figure 7.3 are shown; the other interfaces are identical.

other read-buffers; this can easily be achieved by translating the transfer-requests appropriately.

The core idea, then, is to observe that the new repository and the old repository only differ by attaching a converter at interface W. We assign a new name to this resource and define **aTRep** = relay$^{\mathtt{W}}$**aRep** (and analogously **ā̄TRep** := relay$^{\mathtt{W}}$**ā̄Rep**) with a converter relay that always remembers the existing assignments of reader to writer interfaces and on a transfer-query for two reader interfaces, it simply connects the corresponding writer-interface. The resource **aTRep** is additionally formally described in Figure 7.7.

The converter. Converter relay distinguishes two types of inputs: transfer commands from a writer to a reader TRANSFER(X, Y) are forwarded to the connected repository. Transfer commands between two readers, TRANSFER($\mathtt{R}_1, \mathtt{R}_2$) are translated to transfer commands TRANSFER(X, $\mathtt{R}_2$), where X denotes the writer interface where the value readable at $\mathtt{R}_1$ was first input.

A simple black-box construction. Any protocol that constructs **aRep** from **Rep** (and **Dist**) also constructs relay$^{\mathtt{W}}$**aRep** from relay$^{\mathtt{W}}$**Rep** (and **Dist**), where the assumed resource relay$^{\mathtt{W}}$**Rep** is an insecure repository that also allows information transfer between two receivers, i.e.,

sending a signature from one receiver to another. This is easy to see: assume there was a distinguisher $\mathbf{D}$ for systems $\mathsf{sim}^{\mathtt{E}}\mathsf{relay}^{\mathtt{W}}\mathbf{aRep}$ and $[\mathsf{relay}^{\mathtt{W}}\mathbf{Rep}, \mathbf{Dist}]$, and we are going to construct a distinguisher $\mathbf{D}'$ for the underlying two resources without the converter relay attached. (Note that sim is the same simulator as in Theorem 7.3.3.) Distinguisher $\mathbf{D}'$ simply behaves as $\mathbf{D}$ but additionally emulates relay for queries at the free interface.

This concludes the study of our new model for digital signature schemes.

Part IV

Blockchain Protocols

Chapter 8

A Composable Model for Bitcoin

8.1 Introduction

Since Nakamoto first proposed Bitcoin as a decentralized cryptocurrency [Nak08], several works have focused on analyzing and/or predicting its behavior under different attack scenarios [BDOZ11, ES18, Eya15, Zoh15, SZ15, KKKT16, PS17]. However, a core question remained:

> What security goal does Bitcoin achieve under what assumptions?

An intuitive answer to this question was already given in Nakamoto's original white paper [Nak08]: Bitcoin aims to achieve some form of consensus on a set of valid transactions. The core difference of this consensus mechanism with traditional consensus [LSP82, Lam98, Lam02, Rab83] is that it does not rely on having a known (permissioned) set of participants, but everyone can join and leave at any point in time. This is often referred to as the *permissionless* model. Consensus in this model is achieved by shifting from the traditional assumptions on the fraction of cheating versus honest participants, to assumptions on the collective computing power of the cheating participants compared to the total computing power of the parties that support the consensus

mechanism. The core idea is that in order for a party's action to affect the system's behavior, it needs to prove that it is investing sufficient computing resources. In Bitcoin, these resources are measured by means of solutions to a presumably computation-intensive problem.

Although the above idea is implicit in [Nak08], a formal description of Bitcoin's goal had not been proposed or known to be achieved (and under what assumptions) until the recent works of Garay, Kiayias, and Leonardos [GKL15] and Pass, Seeman, and shelat [PSS17]. In a nutshell, these works set forth models of computation and, in these models, an abstraction of Bitcoin as a distributed protocol, and proved that the output of this protocol satisfies certain security properties, for example the *common prefix* [GKL15] or consistency [PSS17] property. This property confirms—under the assumption that not too much of the total computing power of the system is invested in breaking it—a heuristic argument used by the Bitcoin specification: if some block makes it deep enough into the blockchain of an honest party, then it will eventually make it into the blockchain of every honest party and will never be reversed.[1] In addition to the common prefix property, other quality properties of the output of the abstracted blockchain protocol were also defined and proved. A more detailed description of the security properties and a comparison of the assumptions in [GKL15] and [PSS17] is included in Section 8.7.1.

8.1.1 Bitcoin: A Service for Cryptographic Protocols

Evidently, the main use of the Bitcoin protocol is as a decentralized monetary system with a payment mechanism, which is what it was designed for. And although the exact economic forces that guide its sustainability are still being researched, and certain rational models predict it is not a stable solution, it is a fact that Bitcoin has not met any of these pessimistic predictions for several years and it is not clear it ever will do. And even if it does, the research community has produced and is testing several alternative decentralized cryptocurrencies, e.g., [MGGR13, SCG+14, But13], that are more functional and/or resilient to theoretic attacks than Bitcoin. Thus, it is reasonable to assume that decentralized cryptocurrencies are here to stay.

[1] In the original Bitcoin heuristic "deep enough" is defined as six blocks, whereas in these works it is defined as linear in an appropriate security parameter.

This leads to the natural questions of how one can use this new reality to improve the security and/or efficiency of cryptographic protocols. First answers to this question were given in [ADMM14, ADMM16, BK14, KVV16, KB16, KMB15, KB14, AD15] where it was shown how Bitcoin can be used as a punishment mechanism to incentivize honest behavior in higher level cryptographic protocols such as fair lotteries, poker, and general multi-party computation.

But in order to formally define and prove the security of the above constructions in a widely accepted cryptographic framework for multi-party protocols, one needs to define what it means for these protocols to be run in a world that gives them access to the Bitcoin network as a resource to improve their security. In other words, the question now becomes:

> What functionality can Bitcoin provide to cryptographic protocols?

To address this question, Bentov and Kumaresan [BK14] introduced a model of computation in which protocols can use a punishment mechanism to incentivize adversaries to adhere to their protocol instructions. As a basis, they use the universal composition framework of Canetti [Can01b], but the proposed modifications do not support composition and it is not clear how standard UC cryptographic protocols can be cast as protocols in that model.

In a different direction, Kiayias, Zhou, and Zikas [KZZ16] connected the above question with the original question of Bitcoin's security goal. More concretely, they proposed identifying the resource that Bitcoin (or other decentralized cryptocurrencies) offers to cryptographic protocols as its security goal, and expressing it in a standard language compatible with the existing literature on cryptographic mulit-party protocols. More specifically, they modeled the ideal guarantees as a transaction-ledger functionality in the universal composition framework. To be more precise, the ledger of [KZZ16] is formally a global setup in the (extended) GUC framework of Canetti et al. [CDPW07b].

In a nutshell, the ledger proposed by [KZZ16] corresponds to a trusted third party which keeps a state of blocks of transactions and makes it available, upon request, to any party. Furthermore, it accepts messages/transactions from any party and records them as long as they pass

an appropriate validation procedure that depends on the above publicly available state as well as other registered messages. Periodically, this ledger puts the transactions that were recently registered into a block and adds them into the state. The state is available to everyone. As proved in [KZZ16], giving multi-party protocols access to such a transaction-ledger functionality allows for formally capturing, within the composable (G)UC framework, the mechanism of leveraging security loss with coins. The proposed ledger functionality guarantees in an ideal manner all properties that one could expect from Bitcoin and encompasses the properties in [GKL15, PSS17]. Therefore, it is natural to postulate that it is a candidate for defining the security goal of Bitcoin (and potentially other decentralized cryptocurrencies). However, the ledger functionality proposed by [KZZ16] was not accompanied by a security proof that any of the known cryptocurrencies implements it.

However, as we show, despite being a step in the right direction, the ledger proposed in [KZZ16] cannot be realized under standard assumptions about the Bitcoin network. On the positive side, we specify a new transaction ledger functionality which still guarantees all properties postulated in [GKL15, PSS17], and prove that a reasonable abstraction of the Bitcoin protocol implements this ledger. In our construction, we describe Bitcoin as a UC protocol which generalizes both the protocols proposed in [GKL15, PSS17]. Along the way we identify the assumptions in each of [GKL15, PSS17] by devising a compound way of capturing such assumptions in UC, which enables us to compare their strengths.

8.1.2 Our Contributions

We put forth the first universally composable (simulation-based) proof of security of Bitcoin in the (G)UC model of Canetti et al. [CDPW07b]. We observe that the ledger functionality proposed by Kiayias et al. [KZZ16] is too strong to be implemented by the Bitcoin protocol—in fact, by any protocol in the permissionless setting, which uses network assumptions similar to Bitcoin. Intuitively, the reason is that the functionality allows too little interference of the simulator with its state, making it impossible to emulate adversarial attacks that result, e.g., in the adversary inserting only transactions coming from parties it wants or that result in parties holding chains of different length.

Therefore, we propose an alternative ledger functionality which shares

certain design properties with the proposal in [KZZ16] but which can be provably implemented by the Bitcoin protocol. The ledger is parametrized by a set of parameters, for example by a generic transaction validation predicate which enables it to capture decentralized blockchain protocols beyond Bitcoin. Our functionality allows for parties/miners to join and leave the computation and allows for adaptive corruption.

We formally prove for which choice of parameters the proposed ledger functionality is implemented by Bitcoin under the assumption that miners which deviate from the Bitcoin protocol do not control a majority of the total hashing power at any point. To this end, we describe an abstraction of the Bitcoin protocol as a UC protocol. Casting Bitcoin in UC allows to precisely model the protocol assumptions, for example knowledge of the network delay or number of hash-function calls per round. We model Bitcoin to work over a network which basically consists of bounded-delay channels. We explain how such a network could be implemented by running the message-diffusion mechanism of the Bitcoin network (which is run over a lower level network of unicast channels). Intuitively, this network is built by every miner, upon joining the system, choosing some existing miners of its choice to use them as relay-nodes. Similar to the protocol in [PSS17], the miners are not aware of (an upper bound on) the actual delay that the network induces. As we argue, this is a strictly weaker model assumption than assuming that the network delay is publicly known such as in [GKL15] (cf. Section 8.2.2).

Our security proof proposes a useful modularization of the Bitcoin protocol. Concretely, we first identify the part of the Bitcoin code which intuitively corresponds to the lottery aspect, provide an ideal UC functionality that reflects this lottery aspect, and prove that this part of the Bitcoin code realizes the proposed functionality. We then analyze the remainder of the protocol in the simpler world where the respective code that implements the lottery aspect is replaced by invocations of the corresponding functionality. Using the UC composition theorem, we can then immediately combine the two parts into a proof of the full protocol.

As is the case with the so-called *backbone* protocol from [GKL15] our above UC protocol description of Bitcoin relies only on proofs of work and not on digital signatures. As a result, it implements a somewhat weaker ledger, which does not guarantee that transactions submitted by honest

parties will eventually make it into the blockchain.[2] As a last result, we show that (similarly to [GKL15]) by incorporating public-key cryptography, i.e., taking signatures into account in the validation predicate, we can implement a stronger ledger that ensures that transactions issued by honest users—i.e., users who do not sign contradicting transactions and who keep their signing keys for themselves—are guaranteed to be eventually included into the blockchain. The fact that our protocol is described in UC makes this a straight-forward, modular construction using the proposed transaction ledger as a hybrid. In particular, we do not need to consider the specifics of the Bitcoin protocol in the proof of this step. This also allows us to identify the maximum (worst-case) delay a user needs to wait before being guaranteed to see its transaction on the blockchain and be assured that it will not be inverted.

8.1.3 Overview of Bitcoin and Related Work

High-level introduction. At a high level, the Bitcoin protocol works as follows: The parties (also referred to as *miners*) collect and circulate messages (transactions) from users of the network, check that they satisfy some commonly agreed validity property, put the valid transactions into a block, and then try to find appropriate metadata such that the hash of the block-contents and this metadata is of a specific form—concretely that, parsed as a binary string, it has a sufficient number of leading zeros. This is often referred to as a solving a mining puzzle and the intuition behind it is that the best strategy for finding such metadata is supposedly by trial-and-error. Thus, informally, the probability that some party finds appropriate metadata increases proportional to the number of times some party attempts a hash computation. And the more leading zeros we require from a correct puzzle solution the harder it is to find one, since the solution space of the puzzle is smaller.

Intuitively, a successful solution can be seen as a *proof-of-work* (POW) that testifies to the fact that the miner presenting has in fact tried a large number of hash queries. Once a miner finds such a solution, he puts it into a block and sends it to the other miners. The miners who receive it check that it satisfies some validity property (see below) and if so create new metadata using the hash of this (newly minted) block and put this

[2] We formulate a weakened guarantee, which we then amplify using digital signatures.

metadata together with transactions that are still valid into a new block and start working on solving the puzzle induced by this block. Since a block is rendered valid by a miner only if it includes a hash-pointer to a previous valid block in the view of this miner, the view consists of a set of linked lists, namely a sequence of valid blocks each with a hash-pointer to its predecessor in the list. Each such list is called a blockchain or simply chain. All lists have a common starting point which is the so-called genesis block of Bitcoin. Hence, the entire view of a miner could be modeled as a tree, where the root is the genesis block, the nodes are valid blocks, and the hash-pointers correspond to (directed) edges.

The works of Garay, Kiayias, and Leonardos [GKL15] and that of Pass, Seeman, and shelat [PSS17] contain the first formal specifications and security proofs of the Bitcoin protocol. The proved security in these works is property-based. They prove that conditioned on the largest part of the network following the Bitcoin protocol (in fact an abstraction and generalization thereof), the output of this so-called backbone protocol satisfies three properties with overwhelming probability. We only informally describe these properties here. We will meet their formalization when analyzing the Bitcoin protocol in UC. In the following, let $t_1 \leq t_2$ be two points in time during the protocol execution.

- *Common prefix*: Any two valid chains $\mathcal{C}_{t_1}$, $\mathcal{C}_{t_2}$ adopted by some honest parties at times t_1 and t_2, respectively, share a large common prefix. This is typically quantified by specifying a value k (the common-prefix parameter) and the size of the common prefix is required to be at least $|\mathcal{C}_{t_1}| - k$.

- *Chain growth*: For time-intervals $[t_1, t_2]$ of reasonable extent, the increase in number of blocks —measured as the difference between any two valid chains $\mathcal{C}_{t_1}$ and $\mathcal{C}_{t_2}$ adopted by some honest parties at times t_1 and t_2, respectively — is guaranteed to be substantial. The relationship between time and chain-length is typically referred to as the chain-growth coefficient.

- *Chain quality*: For any honest party and its adopted valid chain $\mathcal{C}_t$ at time t, it holds that any consecutive sequence of blocks of reasonable extent in $\mathcal{C}_t$ is guaranteed to contain blocks contributed by honest parties. The proportion of honestly mined blocks is typically refereed to as the chain-quality coefficient.

Chain quality and chain growth are often expressed with respect to the common-prefix parameter k. That is, as the fraction of honestly mined blocks in a consecutive sequence of k blocks, and as the time interval within which an increase of k blocks is guaranteed (except with negligible probability in k).

Network assumptions and random oracle. The models put forth in [GKL15] and [PSS17] assume a multicast network—i.e., a network where a party sends messages to arbitrary other parties[3]—and abstract the hash function as a random oracle. Furthermore, they both have an explicit round-based model of execution where parties proceed in rounds. There are some slight differences between the two models. For example, in [GKL15] every party makes q hash-queries (i.e., q RO calls) in each round as opposed to [PSS17] where every party makes one hash-query per round. Second, in [PSS17], the adversary might choose to delay message delivery but the statements are proved assuming no message is delayed by more than Δ rounds — also known as the partial-synchronous setting — while the initial model put forth in by [GKL15] was more synchronous (and was lifted to the partial synchronous model later). We note that since the number of hash-queries is fixed in both models, this implies that parties know exactly in which round they are, as they could simply count the number of queries made to the random oracle (and by definition of their models no party goes to round $r + 1$ before all parties have finished round r). Note that the partial-synchronous protocol execution model in [PSS17] is a *strictly* weaker setting than a synchronous execution model with a fixed delay of one round.

Property-based vs simulation-based security. Proving that Bitcoin satisfies the above properties has been an essential step into the direction of understanding the security goals of Bitcoin. But as argued above, this does not offer the tool to be able to argue security of cryptographic protocols that use Bitcoin—e.g., to achieve an improved fairness notion [ADMM14, ADMM16, BK14, KVV16, KB16, KMB15, KB14, AD15]—without the need to always look at the Bitcoin specifics. In other

[3]Unlike [GKL15] where this operation is referred to as broadcast, we choose to call it multicast here to avoid confusion with the standard broadcast primitive in the Byzantine agreement literature that offers stronger consistency guarantees.

words, such property based security definitions do not support composition. The standard way to allow for such a generic use of blockchain protocols as a cryptographic resource, is to prove that it implements an ideal functionality in a composable framework. Intuitively, in such frameworks, a composition theorem states that we can replace calls to a functionality with invocation of a protocol implementing it without worrying about the protocol's internals.

8.2 Principles of our Model

In this section we describe our (G)UC-based model of execution for the Bitcoin protocol. We remark that providing such a formal model of execution forces us to make explicit all the implicit assumptions from previous works. As we lay down the theoretical framework, we will also discuss these assumptions along with their strengths and differences.

Bitcoin miners are represented as players—formally Interactive Turing Machine instances (ITIs)—in a multi-party computation. For notational convenience, we denote the identities of these machines by P_i, i.e, $P_i = (\mathsf{pid}_i, \mathsf{sid}_i)$ and call P_i a party for short. The index i is used to distinguish two identifiers, i.e., $P_i \neq P_j$ and otherwise carries no meaning. Parties interact which each other by exchanging messages over an unauthenticated multicast network with eventual delivery (see below) and might make queries to a common random oracle. We will assume a central adversary $\mathcal{A}$ who gets to corrupt miners and might use them to attempt to break the protocol's security. As is common in (G)UC, the resources available to the parties are described as hybrid functionalities. Before we provide the formal specification of such functionalities, we first discuss a delicate issue that relates to the set of parties (ITIs) that might interact with an ideal functionality.

8.2.1 Functionalities with Dynamic Party Sets

In many UC functionalities, the set of parties is defined upon initiation of the functionality and is not subject to change throughout the lifecycle of the execution. Nonetheless, UC does provide support for functionalities in which the set of parties that might interact with the functionality is dynamic. In fact, this dynamic nature is an inherent feature of the

Bitcoin protocol—where miners come and go at will. In this work we make this explicit by means of the following mechanism: All the functionalities considered here include the following instructions that allow honest parties to join or leave the set $\mathcal{P}$ of players that the functionality interacts with, and inform the adversary about the current set of registered parties:[4]

- Upon receiving $(\text{REGISTER}, \text{sid})$ from some party P_i (or from $\mathcal{A}$ on behalf of a corrupted P_i), set $\mathcal{P} = \mathcal{P} \cup \{P_i\}$. Return $(\text{REGISTER}, \text{sid}, P_i)$ to the caller.
- Upon receiving $(\text{DE-REGISTER}, \text{sid})$ from some party $P_i \in \mathcal{P}$, the functionality updates $\mathcal{P} := \mathcal{P} \backslash \{P_i\}$ and returns $(\text{DE-REGISTER}, \text{sid}, P_i)$ to P_i.
- Upon receiving $(\text{IS-REGISTERED}, \text{sid})$ from some party P_i, return $(\text{REGISTER}, \text{sid}, b)$ to the caller, where the bit b is 1 if and only if $P_i \in \mathcal{P}$.[5]
- Upon receiving $(\text{GET-REGISTERED}, \text{sid})$ from $\mathcal{A}$, the functionality returns the response $(\text{GET-REGISTERED}, \text{sid}, \mathcal{P})$ to $\mathcal{A}$.

In addition to the above registration instructions, global setups, i.e., shared functionalities that are available both in the real and in the ideal world and allow parties connected to them to share state [CDPW07b], allow also UC functionalities to register with them. Concretely, global setups include, in addition to the above party registration instructions, two registration/de-registration instructions for functionalities:[6]

- Upon receiving $(\text{REGISTER}, \text{sid}_G)$ from a functionality $\mathcal{F}$ (with session-id sid), update $F := F \cup \{(\mathcal{F}, \mathsf{sid})\}$.

[4] Note that making the set of parties dynamic means that the adversary needs to be informed about which parties are currently in the computation so that he can chose how many (and which) parties to corrupt.

[5] Note that typically a party knows whether it is registered at a functionality or not (and in which session). However, it might be useful for another functionality to access this information via the dummy party corresponding to P_i. The exact dynamics of such an information exchange can be found in [CSV16b, Section 2]. This is not to be confused with the fact that functionalities can always communicate with global setups by the standard message exchange mechanism.

[6] Recall that a shared functionality knows the identity of each ITM that calls it, which by definition includes the session identifier.

- Upon receiving $(\text{DE-REGISTER}, \text{sid}_G)$ from a functionality $\mathcal{F}$ (with session-id sid), update $F := F \setminus \{(\mathcal{F}, \mathsf{sid})\}$.

- Upon receiving $(\text{GET-REGISTERED-F}, \text{sid}_G)$ from $\mathcal{A}$, reply to $\mathcal{A}$ with $(\text{GET-REGISTERED-F}, \text{sid}_G, F)$.

We use the expression sid_G to refer to the encoding of the session identifier of global setups. By default (and if not otherwise stated), the above four (or seven in case of global setups) instructions will be part of the code of *all* ideal functionalities considered in this work. However, to keep the description simpler we will omit these instructions from the formal descriptions unless deviations are defined.

8.2.2 Modeling Network Assumptions

In many situations, one cannot tolerate a complete asynchronous network such as the standard UC communication mechanism. For example, we want to argue about liveness properties of blockchains, which requires communication with eventual delivery guarantees as time goes by (see below how we model time). We describe such a network based on ideas from from [KMTZ13, BHMQU05, CGHZ16]. In particular, we capture such communication by a network functionality $\mathcal{F}_{\text{N-MC}}^{\Delta}$ that provides each party or miner $P_s \in \mathcal{P}$ the capability to multicast a message. For every newly sent message, say m, the network functionality creates a unique identifier mid for each triple (P_j, P_j, m), where $P_j \in \mathcal{P}$ is a potential receiver. This handle is needed to succinctly refer to a message circulating in the network in a fine-grained manner. The network does not provide any information to any receiver about who else is using it or where a message originates from. More precisely, messages are buffered but the information of who is the sender is not provided to any recipient.

The adversary—who is informed about both the content of the messages and about the handles—is allowed to delay messages by any finite amount, and allowed to deliver them in an arbitrary out-of-order manner. To ensure that the adversary cannot arbitrarily delay the delivery of messages submitted by honest parties, we use the following idea: The network works in a "fetch message" mode, which means that parties need to actively query for the message (for example, a party can query for messages once in a round). If the adversary wishes to delay the delivery of

some message with message ID mid, he needs to submit an integer value T_{mid}—the *delay* for the message-in-transmission with identifier mid. For example, if mid refers to the triple (P_s, P_j, m), this will have the effect that only after the next T_{mid} fetch attempts by P_j, P_j will be able to report the receipt of this particular message m. Importantly, the network does not accept more than Δ accumulative delay for any mid. To allow the adversary freedom in scheduling the delivery of messages, we allow him to input delays more than once, which are added to the current delay amount. If the adversary wants to deliver the message in the next activation, all he needs to do is submit a negative delay. Furthermore, we allow the adversary to schedule more than one messages to be delivered in the same "fetch" command. Finally, to ensure that the adversary is able to re-order such batches of messages arbitrarily, we allow $\mathcal{A}$ to send special $(\text{SWAP}, \text{mid}, \text{mid}')$ commands that have as an effect to change the order of the corresponding messages. Last but not least, the adversary is further allowed to do partial and inconsistent multicasts, i.e., where different messages are sent to different parties. This is the main difference of such a multicast network from a broadcast network. The description appears in Figure 8.1.

From unicast to multicast. A natural question is how to get the above multicast network from simpler channels. Note that in Bitcoin, parties/miners communicate over an incomplete network and a standard diffusion mechanism is employed: The sender sends the message it wishes to multicast to all its neighbors who check that a message with the same content was not received before, and if this is the case forward it to their neighbors, who then do the same check, and so on.

In fact, a multicast network can be built from unicast channels. That is, one essentially assumes for each miner $P_R \in \mathcal{P}$ a channel functionality $\mathcal{F}_{\text{U-CH}}^{\Delta, P_R}$ — which is parameterized by a receiver P_R and an upper bound on the delay Δ — to which any other party $P_i \in \mathcal{P}$ can connect and input messages to be delivered to P_R. A miner connecting to the unicast channel with receiver P_R models the real-world process of looking up P_R (e.g., a public node in the network) and using this party to disseminate future messages. The unicast channel should have some similar properties as the above network, namely:

- They guarantee (reliable) delivery of messages within a delay pa-

Functionality $\mathcal{F}_{\text{N-MC}}^{\Delta}$

The functionality manages the set possible senders and receivers denoted by $\mathcal{P}$. Any newly registered (resp. deregistered) party is added to (resp. deleted from) $\mathcal{P}$. The functionality manages a list $\vec{M}$, initially the empty list.

- *Honest sender multicast:*
  Upon receiving $(\text{MULTICAST}, \text{sid}, m)$ from some $P_s \in \mathcal{P}$, where $\mathcal{P} = \{P_1, \ldots, P_n\}$ denotes the current party set, do:
  1. Choose n new unique message-IDs $\text{mid}_1, \ldots, \text{mid}_n$,
  2. Define $2n$ new variables $D_{\text{mid}_1} := D_{\text{mid}_1}^{MAX} \ldots := D_{\text{mid}_n} := D_{\text{mid}_n}^{MAX} := 1$,
  3. Set $\vec{M} := \vec{M}||(m, \text{mid}_1, D_{\text{mid}_1}, P_1)|| \ldots ||(m, \text{mid}_n, D_{\text{mid}_n}, P_n)$,
  4. Send $(\text{MULTICAST}, \text{sid}, m, P_s, (P_1, \text{mid}_1), \ldots, (P_n, \text{mid}_n))$ to the adversary.

- *Adversarial sender (partial) multicast:*
  Upon receiving $(\text{MULTICAST}, \text{sid}, (m_{i_1}, P_{i_1}), \ldots, (m_{i_\ell}, P_{i_\ell})$ from the adversary with $\{P_{i_1}, \ldots, P_{i_\ell}\} \subseteq \mathcal{P}$, do:
  1. Choose ℓ new unique message-IDs $\text{mid}_{i_1}, \ldots, \text{mid}_{i_\ell}$,
  2. initialize ℓ new variables $D_{\text{mid}_{i_1}} := D_{\text{mid}_{i_1}}^{MAX} := \ldots := D_{\text{mid}_{i_\ell}} := D_{\text{mid}_{i_\ell}}^{MAX} := 1$,
  3. set $\vec{M} := \vec{M}||(m_{i_1}, \text{mid}_{i_1}, D_{\text{mid}_{i_1}}, P_{i_1})|| \ldots ||(m_{i_\ell}, \text{mid}_{i_\ell}, D_{\text{mid}_{i_\ell}}, P_{i_\ell})$,
  4. send $(\text{MULTICAST}, \text{sid}, (m_{i_1}, P_{i_1}, \text{mid}_{i_1}), \ldots, (m_{i_\ell}, P_{i_\ell}, \text{mid}_{i_\ell})$ to the adversary.

- *Honest party fetching:*
  Upon receiving $(\text{FETCH}, \text{sid})$ from $P_i \in \mathcal{P}$ (or from $\mathcal{A}$ on behalf of P_i if P_i is corrupted):
  1. For all tuples $(m, \text{mid}, D_{\text{mid}}, P_i) \in \vec{M}$, set $D_{\text{mid}} := D_{\text{mid}} - 1$.
  2. Let $\vec{M}_0^{P_i}$ denote the subvector $\vec{M}$ including all tuples of the form $(m, \text{mid}, D_{\text{mid}}, P_i)$ with $D_{\text{mid}} = 0$ (in the same order as they appear in $\vec{M}$). Delete all entries in $\vec{M}_0^{P_i}$ from $\vec{M}$, and send $\vec{M}_0^{P_i}$ to P_i.

- *Adding adversarial delays:*
  Upon receiving $(\text{DELAYS}, \text{sid}, (T_{\text{mid}_{i_1}}, \text{mid}_{i_1}), \ldots, (T_{\text{mid}_{i_\ell}}, \text{mid}_{i_\ell}))$ from the adversary do the following for each pair $(T_{\text{mid}_{i_j}}, \text{mid}_{i_j})$:
  If $D_{\text{mid}_{i_j}}^{MAX} + T_{\text{mid}_{i_j}} \leq \Delta$ and mid is a message-ID registered in the current $\vec{M}$, set $D_{\text{mid}_{i_j}} := D_{\text{mid}_{i_j}} + T_{\text{mid}_{i_j}}$ and set $D_{\text{mid}_{i_j}}^{MAX} := D_{\text{mid}_{i_j}}^{MAX} + T_{\text{mid}_{i_j}}$; otherwise, ignore this pair.

- *Adversarially reordering messages:*
  Upon receiving $(\text{SWAP}, \text{sid}, \text{mid}, \text{mid}')$ from the adversary, if mid and mid' are message-IDs registered in the current $\vec{M}$, then swap the triples $(m, \text{mid}, D_{\text{mid}}, \cdot)$ and $(m, \text{mid}', D_{\text{mid}'}, \cdot)$ in $\vec{M}$. Return $(\text{SWAP}, \text{sid})$ to the adversary.

Figure 8.1: The network functionality with eventual delivery guarantees. Note that for a list $\vec{M}$ we denote by the symbol $||$ the operation which appends a new element to $\vec{M}$.

rameter but are otherwise specified to be of asynchronous nature (see below) and hence no protocol can rely on timings regarding the delivery of messages. The adversary might delay any message sent through such a channel, but at most by Δ. In particular, the adversary cannot block messages. However, he can induce an arbitrary order on the messages sent to some party.

- The receiver gets no information other than the messages themselves. In particular, a receiver cannot link a message to its sender nor can he observe whether or not two messages were sent from the same sender.

- The channel offers no privacy guarantees. The adversary is given read access to all messages sent on the network.

In Appendix B.1, we provide this channel functionality for completeness and explain how a simple round-based diffusion mechanism can be used to implement a multicast mechanism from unicast channels as long as the corresponding network among honest parties stays strongly connected. (A network graph is strongly connected if there is a directed path between any two nodes in the network, where the unicast channels are seen as the directed edges from sender to receiver.)

On functionally black-box use of the network. A key difference between the initial model of [GKL15] and [PSS17] was that in the latter the parties do not know any bound on the delay of the network. In particular, although both models are in the synchronous setting, in [PSS17] and in the extended model provided in [GKL15], a party in the protocol does not know when to expect a message which was sent to it in the previous round. Using terminology from [Ros12], the protocol uses the channel in a *functionally black-box* manner. Restricting to such protocols—a restriction which we also adopt in this work—is in fact implying a weaker assumption on the protocol than standard (known) bounded-delay channel. Intuitively the reason is that no such protocol can realize a bounded-delay network with a known upper bound (unless it sacrifices termination) since the protocol cannot decide whether or not the bound has been reached.

8.2.3 Modeling Time and Clock-dependent Protocol Execution

Katz et al. [KMTZ13], proposed a methodology for casting synchronous protocols in UC by assuming they have access to an ideal functionality $\mathcal{G}_{\text{CLOCK}}$, *the clock*, that allows parties to ensure that they proceed in synchronized rounds. Informally, the idea is that the clock keeps track of a round variable whose value the parties can request by sending it $(\text{CLOCK-READ}, \text{sid}_C)$. This value is updated only once all honest parties sent the clock a $(\text{CLOCK-UPDATE}, \text{sid}_C)$ command. We lift their idea to a shared setup. The global clock functionality $\mathcal{G}_{\text{CLOCK}}$ is a shared clock that may interact with more than one protocol session. The global clock provides a means for parties to synchronize each of their sessions.[7] The clock can also be used as a local (not shared) hybrid functionality, in which case the number of sessions it will synchronize is simply one. The description is given in Figure 8.2.

Given a clock, the authors of [KMTZ13] describe how synchronous protocols can maintain their necessary round structure in UC: For every round ρ each party first executes all its round-ρ instructions and then sends the clock a CLOCK-UPDATE command. Subsequently, whenever activated, it sends the clock a CLOCK-READ command and does not advance to round $\rho + 1$ before it sees the clocks variable being updated. This ensures that no honest party will start round $\rho + 1$ before every honest party has completed round ρ. In [KZZ16], this idea was transfered to the (G)UC setting, by assuming that the clock is a global setup. This allows for different protocols to use the same clock and is the model we will also use here.

As argued in [KMTZ13], in order for an eventual-delivery (aka guaranteed termination) functionality to be UC implementable by a synchronous protocol it needs to keep track of the number of activations that an honest party gets—so that it knows when to generate output for honest parties. This requires that the protocol itself, when described as a UC interactive Turing-machine instance (ITI), has a predictable behavior when it comes to the pattern of activations that it needs before it sends the clock an update command. We capture this property in a generic

[7] The functionality presented here is different from shared clock functionalities used in prior work. We believe that this version here is closer to the spirit of the GUC/EUC version of UC.

Functionality $\mathcal{G}_{\text{CLOCK}}$

The functionality manages the set $\mathcal{P}$ of registered identities, i.e., parties $P = (\mathsf{pid}, \mathsf{sid})$. It also manages the set F of functionalities (together with their session identifier). Initially, $\mathcal{P} := \emptyset$ and $F := \emptyset$.

For each session sid the clock maintains a variable τ_{sid}. For each identity $P := (\mathsf{pid}, \mathsf{sid}) \in \mathcal{P}$ it manages variable d_P. For each pair $(\mathcal{F}, \mathsf{sid}) \in F$ it manages variable $d_{(\mathcal{F},\mathsf{sid})}$ (all integer variables are initially 0).

Synchronization:

- Upon receiving $(\text{CLOCK-UPDATE}, \text{sid}_C)$ from some party $P \in \mathcal{P}$ set $d_P := 1$; execute *Round-Update* and forward $(\text{CLOCK-UPDATE}, \text{sid}_C, P)$ to $\mathcal{A}$.
- Upon receiving $(\text{CLOCK-UPDATE}, \text{sid}_C)$ from some functionality $\mathcal{F}$ in a session sid such that $(\mathcal{F}, \mathsf{sid}) \in F$ set $d_{(\mathcal{F},\mathsf{sid})} := 1$, execute *Round-Update* and return $(\text{CLOCK-UPDATE}, \text{sid}_C, \mathcal{F})$ to this instance of $\mathcal{F}$.
- Upon receiving $(\text{CLOCK-READ}, \text{sid}_C)$ from any participant (including the environment on behalf of a party, the adversary, or any ideal—shared or local—functionality) return $(\text{CLOCK-READ}, \text{sid}_C, \tau)$ to the requestor.

Procedure Round-Update: For each session sid do: If $d_{(\mathcal{F},\mathsf{sid})} := 1$ for all $\mathcal{F} \in F$ and $d_P = 1$ for all honest parties $P = (\cdot, \mathsf{sid}) \in \mathcal{P}$, then set $\tau_{\mathsf{sid}} := \tau_{\mathsf{sid}} + 1$ and reset $d_{(\mathcal{F},\mathsf{sid})} := 0$ and $d_P := 0$ for all parties $P = (\cdot, \mathsf{sid}) \in \mathcal{P}$.

Figure 8.2: The shared/global clock functionality. We assume lazy creation of variables, i.e., a variable is only created once it is needed.

manner in Definition 8.2.1.

To follow the definition recall the mechanics of activations in UC. In a UC protocol execution, an honest party (ITI) gets activated either by receiving an input from the environment, or by receiving a message from one of its hybrid-functionalities (or from the adversary). Any activation results in the activated ITI performing some computation on its view of the protocol and its local state and ends with either the party sending a message to some of its hybrid functionalities or sending an output to the environment, or not sending any message. In either of these cases, the party loses the activation.[8]

For any given protocol execution, we define the *honest-input sequence* $\vec{\mathcal{I}}_H$ to consist of all inputs that the environment gives to honest parties in

[8] In the latter case the activation goes to the environment by default.

the given execution (in the order that they were given) along with the identity of the party who received the input. For an execution in which the environment has given m inputs to the honest parties in session sid in total, $\vec{\mathcal{I}}_H$ is a vector of the form $((x_1, id_1), \ldots, (x_m, id_m))$, where x_i is the i-th input that was given in this execution, and id_i is the corresponding identity (i.e., $id_i = (\mathsf{pid}_i, \mathsf{sid})$ for some bitstring pid) that received this input in this session. We further define the *timed honest-input sequence,* denoted as $\vec{\mathcal{I}}_H^T$, to be the honest-input sequence augmented with the respective clock time when an input was given. If the timed honest-input sequence of an execution is $\vec{\mathcal{I}}_H^T = ((x_1, id_1, \tau_1), \ldots, (x_m, id_m, \tau_m))$, this means that $((x_1, id_1), \ldots, (x_m, id_m))$ is the honest-input sequence corresponding to this execution, and for each $i \in [n]$, τ_i is the time of the global clock when input x_i was handed to id_i.

Definition 8.2.1. A $\mathcal{G}_{\text{CLOCK}}$-hybrid protocol Π has a *predictable synchronization pattern* iff there exist an algorithm $\mathsf{predict\text{-}time}_\Pi(\cdot)$ such that for any possible execution of Π in a session sid (i.e., for any adversary and environment, and any choice of random coins) the following holds: If $\vec{\mathcal{I}}_H^T = ((x_1, id_1, \tau_1), \ldots, (x_m, id_m, \tau_m))$ is the corresponding timed honest-input sequence for this session, then for any $i \in [m-1]$:

$$\mathsf{predict\text{-}time}_\Pi((x_1, id_1, \tau_1), \ldots, (x_i, id_i, \tau_i)) = \tau_{i+1},$$

where τ_{i+1} is the clock time for this session (cf. Figure 8.2).

As we argue, all synchronous protocol described in this work are designed to have a predictable synchronization pattern.

8.2.4 Modeling Hash Queries

As usual in cryptographic proofs, the queries to the hash function are modeled by assuming access to a random oracle (functionality) $\mathcal{F}_{\text{RO}}$. This functionality is specified as follows: upon receiving a query $(\text{EVAL}, \mathsf{sid}, x)$ from a registered party, if x has not been queried before, a value y is chosen uniformly at random from $\{0,1\}^\kappa$ (for security parameter κ) and returned to the party (and the mapping (x, y) is internally stored). If x has been queried before, the corresponding y is returned. The description appears in Figure 8.3.

Functionality $\mathcal{F}_{\text{RO}}$

The functionality is parametrized by the security parameter κ. It maintains the set of registered parties/miners $\mathcal{P}$ (initially set to $\emptyset$) and a (dynamically updatable) function table $\mathcal{T}$ (initially $\mathcal{T} = \emptyset$). For simplicity we write $T[x] = \perp$ to denote the fact that no pair of the form $(x, \dot{)}$ is in $\mathcal{T}$.

- Upon receiving $(\text{EVAL}, \text{sid}, x)$ from some party $P \in \mathcal{P}$ (or from $\mathcal{A}$ on behalf of a corrupted P), do the following:
  1. If $H[x] = \perp$ sample a value y uniformly at random from $\{0,1\}^\kappa$, set $H[x] \leftarrow y$ and add $(x, T[x])$ to $\mathcal{T}$.
  2. Return $(\text{EVAL}, \text{sid}, x, H[x])$ to the requestor.

Figure 8.3: The random oracle functionality.

A note on global random oracles and PoW. In our model, the random oracle is a local setup. In fact, abstracting hash-queries as calls to a global random oracle (GRO) runs into intrinsic problems in the PoW-setting because of two reasons: (1) at an intuitive level this would imply that the environment could make queries to the GRO and then provide them to the adversary. As such, no real restriction on the adversary exists; (2) at the more technical level, the non-programmability of the GRO forces the simulator to create blocks that indeed carry sufficient work. Since the simulator needs to also simulate the hash queries of honest parties, this would only be feasible if he had a much larger query budget than the real-world adversary has, which is not possible as the GRO needs to behave identically in the real and ideal world.

8.2.5 Assumptions as UC-Functionality Wrappers

In order to prove statements about cryptographic protocols one often makes assumptions about what the environment (or the adversary) can or cannot do. For example, a standard assumption in [GKL15, PSS17] is that in each round the adversary cannot do more calls to the random oracle than what the honest parties (collectively) can do. This can be captured by assuming a restricted environment and adversary which balances the amount of times that the adversary queries the random oracle. In a property-based treatment such as [GKL15, PSS17] this assumptions is

typically acceptable. Also in a composable model such restrictions can be formulated. However, restricting the environment is not compliant with a general composition theorem.

Therefore, instead of restricting the class of environments/adversaries, we present an alternative approach to capture the fact that the adversary's access to real-world resource is restricted. The general methodology is to capture restrictions by means of a functionality wrapper that wraps the hybrid resources and enforces the restrictions on the adversary by limiting its access to the resource. Such restrictions can become quite complex and we show concrete examples in Section 8.7 to cast the assumptions and derive the equivalent composable statements.

A toy example. To illustrate the general methodology here with an easy example, consider we want to capture a restriction of the adversary's access to the RO. We can easily capture this assumption by means of a functionality wrapper that wraps the RO functionality and enforces a bound on the adversary, for example by assigning to each corrupted party at most q activations per round for some parameter q. To keep track of rounds the functionality registers with the global clock $\mathcal{G}_{\text{CLOCK}}$. For completeness the wrapped random oracle functionality $\mathcal{W}^q(\mathcal{F}_{\text{RO}})$ is found in Figure 8.4.

8.3 The Basic Transaction-Ledger Functionality

The purpose of this section is to describe the basic structure of a ledger functionality $\mathcal{G}_{\text{LEDGER}}$. The presented functionality is very generic in the sense that it is parameterizable by several elements. The idea is that concrete blockchain protocols yield concrete instances of these parameters, while the basic structure, as presented here, remains the same and can be seen as the greatest common divisor of any such blockchain protocol proposal. The formal description of the functionality is given at the end of this section and the remainder of this section outlines its properties.

Wrapped Functionality $\mathcal{W}^q(\mathcal{F}_{\mathrm{RO}})$

The wrapper functionality is parametrized by an upper bound q which restricts the $\mathcal{F}$-evaluations of each corrupted party per round. The functionality manages the variable `counter` and the current set of corrupted miners $\mathcal{P}'$. For each party $P \in \mathcal{P}'$ it manages variables $\texttt{count}_P$.

Initially, $\mathcal{P}' = \emptyset$ and $\texttt{counter} = 0$.

General:

- The wrapper does not interact with the adversary as soon as the adversary tries to exceed its budget of q queries per corrupted party. Registration-queries and their replies are simply relayed without modifications.

Relaying inputs to the random oracle:

- Upon receiving $(\textsc{eval}, \mathrm{sid}, x)$ from $\mathcal{A}$ on behalf of a corrupted party $P \in \mathcal{P}'$, then first execute *Round Reset*. Then, set $\texttt{count}_P \leftarrow \texttt{count}_P + 1$ and only if $\texttt{count}_P \leq q$ forward the request to $\mathcal{F}_{\mathrm{RO}}$ and return to $\mathcal{A}$ whatever $\mathcal{F}_{\mathrm{RO}}$ returns.

- Any other request from any participant or the adversary is simply relayed to the underlying functionality without any further action and the output is given to the destination specified by the hybrid functionality.

Standard UC Corruption Handling:

- Upon receiving $(\textsc{corrupt}, \mathrm{sid}, P)$ from the adversary, set $\mathcal{P}' \leftarrow \mathcal{P}' \cup \{P\}$. If P has already issued $t > 0$ random oracle queries in this round, set $\texttt{count}_P \leftarrow t$. Otherwise set $\texttt{count}_P \leftarrow 0$.

Procedure Round-Reset:
Send $(\textsc{clock-read}, \mathrm{sid}_C)$ to $\mathcal{G}_{\textsc{clock}}$ and receive $(\textsc{clock-read}, \mathrm{sid}_C, \tau)$ from $\mathcal{G}_{\textsc{clock}}$. If $|\tau - \texttt{counter}| > 0$ and the new time τ is even (i.e., a new round started), then set $\texttt{count}_P := 0$ for each participant $P \in \mathcal{P}'$ and set $\texttt{counter} \leftarrow \tau$.

Figure 8.4: The wrapped random oracle.

8.3.1 Introduction and Overview

Our ledger is parametrized by certain algorithms/predicates that allow us to capture a more general version of a ledger which can be instantiated by various cryptocurrencies. Since our abstraction of the Bitcoin protocol is in the synchronous model of computation (this is consistent with known approaches in the cryptographic literature), our ledger is also designed for this synchronous model. Nonetheless, several of our modeling choices are made with the foresight of removing or limiting the use of the clock and leaving room for less synchrony.

At a high level, our ledger $\mathcal{G}_{\text{LEDGER}}$ has a similar structure as the ledger proposed in [KZZ16]. Concretely, anyone (whether an honest miner or the adversary) might submit a transaction which is validated by means of a predicate $\mathsf{Validate}$, and if it is found valid it is added to a buffer $\mathtt{buffer}$. The adversary $\mathcal{A}$ is informed that the transaction was received and is given its contents.[9] Informally, this buffer also contains transactions that, although validated, are not yet deep enough in the blockchain to be considered out-of-reach for an adversary.[10] Periodically, $\mathcal{G}_{\text{LEDGER}}$ fetches some of the transactions in the buffer, and using an algorithm $\mathsf{Blockify}$ creates a block including these transactions and adds this block to its permanent state $\mathtt{state}$, which is a data structure that includes the part of the blockchain the adversary can no longer change. This corresponds to the *common prefix* in [GKL15, PSS17]. Any miner or the adversary is allowed to request a read of the contents of the state.

This sketched specification is simple, but in order to have a ledger that can be implemented by existing blockchain protocols, we need to relax this functionality by giving the adversary more power to interfere with it and influence its behavior. Before sketching the necessary relaxations we discuss the need for a new ledger definition and its potential use as a global setup.

Impossibility to realize the ledger of [KZZ16]. The main reasons why the ledger functionality in [KZZ16] is not realizable by known protocols under reasonable assumptions are as follows: first, their ledger guarantees that parties always obtain the same common state. Even with

[9] This is inevitable since we assume non-private communication, where the adversary sees any message as soon as it is sent, even if the sender and receiver are honest.

[10] E.g., in [KZZ16] the adversary is allowed to permute the contents of the buffer.

strong synchrony assumptions, this is not realizable since an adversary, who just mined a new block, is not forced to inform each party instantaneously (or at all) and thus could for example make parties observe different lengths of the same prefix. Second, the adversarial influence is restricted to permuting the buffer. This is too optimistic, as in reality the adversary can try to mine a new block and possibly exclude certain transactions. Also, this excludes any possibility to quantify quality. Third, letting the update rate be fixed does not adequately reflect the probabilistic nature of Nakamoto-style blockchain protocols.

On the sound usage of a ledger as a global setup. As presented in [KZZ16], a UC ledger functionality $\mathcal{G}_{\text{LEDGER}}$ can be cast as a global setup [CDPW07b] which allows different protocols to share state. This fact holds true for any UC functionality as stated in [CDPW07b] and [CSV16b]. Nonetheless, as pointed out in the recent work of Canetti, Shahaf, and Vald [CSV16b], one needs to be extra careful when replacing a global setup by its implementation, e.g., in the case of $\mathcal{G}_{\text{LEDGER}}$ by the UC Bitcoin protocol. Indeed, such a replacement does not, in general, preserve a realization proof of some ideal functionality $\mathcal{F}$ that is conducted in a ledger-hybrid world, because the simulator in that proof might rely on specific capabilities that are not available any more after replacement (as the global setup is also replaced in the ideal world). The authors of [CSV16b] provide a sufficient condition for such a replacement to be sound. This condition is generally too strong to be satisfied by any natural ledger implementation, which opens the question of devising relaxed sufficient conditions for sound replacements in an MPC context.[11] As this work focuses on the realization of ledger functionalities per se, we can treat $\mathcal{G}_{\text{LEDGER}}$ as a standard UC functionality.

8.3.2 Specific Defining Features

We explain several of the features of the ledger functionality. For an overview of the the parameters and functions we refer to Figure 8.5.

[11] To give an example, a natural condition would be to require that the ideal-world adversary (or simulator) for $\mathcal{F}$ does only use the ledger to submit queries or reading the state, and plays the "dummy adversary" for queries that request the additional adversarial capabilities (i.e., the weaknesses of the ledger). For example, the simulator in [KZZ16] is of this kind.

State-buffer validation

The first relaxation is with respect to the invariant that is enforced by the validation predicate Validate. Concretely, in [KZZ16] it is assumed that the validation predicate enforces that the buffer does not include conflicting transactions, i.e., upon receipt of a transaction, Validate checks that it is not in conflict with the state and the buffer, and if so the transaction is added to the buffer. However, in reality we do not know how to implement such a strong filter, as different miners might be working on different, potentially conflicting sets of transactions.[12] The only time when it becomes clear which of these conflicting transactions will make it into the state is once one of them has been inserted into a block which has made it deep enough into the blockchain (i.e., has become part of `state`). Hence, given that the buffer includes all transactions that might end up in the state, it might at some point include both conflicting transactions.

To enable us for a provably implementable ledger, in this work we take a different approach. The validate predicate will be less restrictive as to which transactions make it into the buffer. Concretely, at the very least, Validate will enforce the invariant that no single transaction in the buffer contradicts the state `state`, while different transactions in `buffer` might contradict each other. Looking ahead, a stronger version that is achievable by employing digital signatures (presented in Section 8.8) could enforce that no submitted transaction contradicts other submitted transactions. As in [KZZ16], whenever a new transaction x is submitted to $\mathcal{G}_{\text{LEDGER}}$, it is passed to Validate which takes as input a transaction and the current state and decides if x should be added to the buffer. Additionally, as `buffer` might include conflicts, whenever a new block is added to the state, the buffer (i.e., every single transaction in `buffer`) is re-validated using Validate and invalid transactions in `buffer` are removed. To allow for this re-validation to be generic, transactions that are added to the buffer are accompanied by certain metadata, i.e., the identity of the submitter, a unique transaction ID txid[13], or the time τ when x was received.

[12] This will be the case for transactions submitted by the adversary even when signatures are used to authenticate transactions.

[13] In Bitcoin, the value txid would be the hash-pointer corresponding to this transaction. Note that the generic ledger can capture explicit guarantees on the ability or disability to link transactions, as this crucially depends on the concrete choice of an ID mechanism.

State update policy and security guarantees

The second relaxation is with respect to the rate and the form and/or origin of transactions that make it into a block. Concretely, instead of assuming that the state is extended in fixed time intervals, we allow the adversary to define when this update occurs. This is done by allowing the adversary, at any point, to propose what we refer to as the next-block candidate `NxtBC`. This is a data structure containing the contents of the next block that $\mathcal{A}$ wants to have inserted into the state. Leaving `NxtBC` empty can be interpreted as the adversary signaling that it does not want the state to be updated in the current clock tick.

Of course allowing the adversary to always decide what makes it into the ledger-state (variable `state`), or if anything ever does, yields a very weak ledger. Intuitively, this would be a ledger that only guarantees the common prefix property [GKL15] but no liveness or chain quality. Therefore, to enable us to capture also stronger properties of blockchain protocols we parameterize the ledger by an algorithm ExtendPolicy that, informally, enforces a state-update policy restricting the freedom of the adversary to choose the next block and implementing an appropriate compliance-enforcing mechanism in case the adversary does not follow the policy. This enforcing mechanism simply returns a default policy-complying block using the current contents of the buffer. We point out that a good simulator for realizing the ledger will avoid triggering this compliance-enforcing mechanism, as this could result in an uncontrolled update of the state which would yield a potential distinguishing advantage. In other words, a good simulator, i.e., ideal-world adversary, always complies with the policy.

In a nutshell, ExtendPolicy takes the current contents of the buffer `buffer`, along with the adversary's recommendation `NxtBC`, and the *block-insertion times vector* $\vec{\tau}_{\texttt{state}}$. The latter is a vector listing the times when each block was inserted into `state`. The output of ExtendPolicy is a vector including the blocks to be appended to the state during the next state-extend time-slot (where again, ExtendPolicy outputting an empty vector is a signal to not extend). To ensure that ExtendPolicy can also enforce properties that depend on who inserted how many (or which) blocks into the state—e.g. the so-called *chain quality* property from [GKL15]—we also pass to it the timed honest-input sequence $\vec{\mathcal{I}}_H^T$ (cf. Section 8.2).

Some examples of how ExtendPolicy allows us to define ways that the

protocol might restrict the adversary's interference in the state-update include the following properties from [GKL15]:

- *Liveness* corresponds to ExtendPolicy enforcing the following policy: If the state has not been extended for more that a certain number of rounds and the simulator keeps recommending an empty `NxtBC`, ExtendPolicy can choose some of the transactions in the buffer (e.g., those that have been in the buffer for a long time) and add them to the next block. Note that a good simulator or ideal-world adversary will never allow for this automatic update to happen and will make sure that he keeps the state extend rate within the right amount.

- *Chain quality* corresponds to ExtendPolicy enforcing the following policy: Every block proposal made by the simulator has to be associated with a special flag hFlag, where intuitively hFlag = 1 indicates that the proposal is generated using the process that an honest miner would follow. ExtendPolicy enforces two things: first, that block proposal indicating hFlag = 1 are frequent enough, and second that such proposals fulfill some specific quality properties (such as including all recent transactions). If these properties are not met, the ledger will define and add a default block to the state. [14] We point out that unlike the original chain-quality property from [GKL15], this policy does not enforce which miner should receive the reward for honest blocks and it is up to the simulator to do so (via the so-called coinbased transaction).[15]

We note that ExtendPolicy is a general concept capable of formulating various properties of blockchain protocols. For example, we can capture that honest (and non-conflicting) transactions eventually make it into the state. Another property could be to formalize that transactions with

[14] More technically, ExtendPolicy looks into the proposed-block sequence and identifies the blocks of `state` that where proposed by the simulator with hFlag set to 1 to deduce how long ago (in time or block-number) the last proposed block that made it into the chain had hFlag = 1.

[15] The actual Bitcoin protocol ensures that at the time when the block was created and circulated in the network the originator of the block was honest. Note that this does not mean that he is still honest when the block makes it into the state *unless* one considers static corruptions only (in which case one can indeed directly argue about the fraction of honest originators in the state). To make this difference is crucial to explicitly see the impact due to adaptive corruptions and was not made explicit in earlier versions of this work.

higher rewards make it into a block faster than others (which we do not consider in this work).

In Section 8.5 we provide the concrete specification of Validate and ExtendPolicy that can be guaranteed for the UC Bitcoin protocol.

Output Slackness and Sliding Window of State Blocks

The common prefix property guarantees that blocks which are sufficiently deep in the blockchain of an honest miner will eventually be included in the blockchain of every honest miner. Stated differently, if an honest miner receives as output from the ledger a state `state`, every honest miner will eventually receive `state` as its output. However, in reality we cannot guarantee that at any given point in time all honest miners see exactly the same blockchain length; this is especially the case when network delays are incorporated into the model, but it is also true in the zero-delay model of [GKL15]. Thus it is unclear how `state` can be defined so that at any point all parties have the same view on it.

Therefore, to have a ledger implementable by standard assumptions we make the following relaxation: We interpret `state` as the view of the state of the miner with the longest blockchain. And we allow the adversary to define for every honest miner P_i a subchain $\texttt{state}_i$ of `state` of length $|\texttt{state}_i| = \texttt{pt}_i$ that corresponds to what P_i gets as a response when he reads the state of the ledger (formally, the adversary can fix a pointer $\texttt{pt}_i$). For convenience, we denote by $\texttt{state}|_{\texttt{pt}_i}$ the subchain of `state` that finishes in the $\texttt{pt}_i$-th block. Once again, to avoid over-relaxing the functionality to an unuseful setup, our ledger allows the adversary to only move the pointers forward and it forbids the adversary to define pointers for honest miners that are too far apart, i.e., more than `windowSize` state blocks. The parameter $\texttt{windowSize} \in \mathbb{N}$ denotes a core parameter of the ledger. In particular, the parameter `windowSize` reflects the similarity of the blockchain to the dynamics of a so-called *sliding window*, where the window of size `windowSize` contains the possible views of honest miners onto `state` and where the head of the window advances with the head of the `state`. In addition, it is convenient to express security properties of concrete blockchain protocols, including the properties discussed above, as assertions that hold within such a sliding window.

Ledger Element	Description
$\mathcal{P}, \mathcal{H}, \mathcal{P}_{DS}$	The party sets and categories: Registered, honest, and honest-but-desynchronized, respectively.
$\vec{\mathcal{I}}_H^T$	The timed honest-input sequence.
predict-time	The function to predict the real-world time advancement.
`state`	The ledger state, i.e., a sequence of blocks containing the content.
`buffer`	The buffer of submitted input values.
$\mathtt{pt}_i$, $\mathtt{state}_i$	The pointer of party P_i into state `state`. This prefix is denoted $\mathtt{state}_i$ for brevity.
$\vec{\tau}_{\mathtt{state}}$	A vector containing for each state block the time when the block added to the ledger state.
τ_L	The current time as reported by the clock.
`NxtBC`	Stores the current adversarial suggestion for extending the ledger state.
Validate	Decides on the validity of a transaction with respect to the current state. Used to clean the buffer of transactions.
ExtendPolicy	The function that specifies the ledger's guarantees in extending the ledger state (e.g., speed, content etc.).
Blockify	The function to format the ledger state output.
`windowSize`	The window size (number of blocks) of the sliding window.
`Delay`	A general delay parameter for the time it takes for a newly joining (after the onset of the computation) miner to become synchronized.

Figure 8.5: Overview of main ledger elements such as parameters and state variables.

Synchrony Aspects and De-Synchronized Parties

In order to keep the ideal execution indistinguishable from the real execution, the adversary should be unable to use the clock for distinguishing. Since in the ideal world when a dummy party receives a CLOCK-UPDATE-message for $\mathcal{G}_{\text{CLOCK}}$ it will forward it, the ledger needs to be responsible that the clock counter does not advance before all honest parties have received sufficiently many activations. This is achieved by the use of the function predict-time$(\vec{\mathcal{I}}_H^T)$ (see Definition 8.2.1), which, as we show, is defined for our ledger protocol. This function allows $\mathcal{G}_{\text{LEDGER}}$ to predict when the protocol would update the round and ensure that it only allows the clock to advance if and only if the protocol would. Observe that the ledger can infer all protocol-relevant inputs/activations to honest parties

and can therefore easily keep track of the honest inputs sequence $\vec{\mathcal{I}}_H^T$. In particular, in global UC communication between the ledger and the (shared) clock functionality is allowed to access the relevant information (namely via a dummy party as defined in [CSV16b]).[16] As the other functions explained above, the function predict-time is a parameter of the (general) ledger functionality and hence needs to be instantiated when realizing a specific ledger such as the Bitcoin ledger (which is the topic of the next section).

A final observation is with respect to guarantees that the protocol (and therefore also the ledger) can give to recently registered honest parties, or to registered parties that get de-registered from the clock (temporarily, for instance). We will call miners *de-synchronized* if one of the above properties are fulfilled for this miner. We denote the set of such miners by $\mathcal{P}_{DS}$.

To provide more intuition, consider the following scenario: An honest party registers as miner in round r and waits to receive from honest parties the transactions to mine and the current longest blockchain. In Bitcoin, upon joining, the miner sends out a special request on the network—we denote this here as a special NEW-MINER-message—and as soon as any party receives it, it responds with the set of transactions and longest blockchain it knows. Due to the network delay Δ, the parties might take up to Δ rounds to receive the NEW-MINER notification, and their response might also take up to Δ rounds before it arrives to the new miner. However, because we do not make any assumption on honest parties knowing Δ they need to start mining as soon as a message arrives (otherwise they might wait indefinitely). But now the adversary, in the worst case, can make these parties mine on any block he wants and have them accept any valid chain he wants as the current state while they wait for the network's response: simply delay everything sent to these parties by honest miners by the maximum delay Δ, and instead, immediately deliver what he wants them to work on. Thus, for the first 2Δ rounds[17]

[16] In order to keep the description below simple, we omit how the ledger exactly infers $\vec{\mathcal{I}}_H^T$, but this is quite straightforward. In particular, the mechanism of [CSV16b] allows to assume that the ledger knows whether a party is registered with the clock or not to deduce whether it is synchronized or de-synchronized.

[17] For technical reasons described in Section 8.4, Δ rounds in the protocol correspond to 2Δ clock-ticks and hence the ledger parameter will concretely be defined as $\texttt{Delay} = 4\Delta$.

these parties are practically in the control of the adversary and their computing power is contributed to his. The ledger parameter `Delay` describes the time it takes for a newly joining party, which joins later than in the very first round, to become officially synchronized.

Functionality $\mathcal{G}_{\text{LEDGER}}$

General: The functionality is parametrized by four algorithms Validate, ExtendPolicy, Blockify, and predict-time, along with two parameters $\texttt{windowSize}, \texttt{Delay} \in \mathbb{N}$. The functionality manages variables `state`, `NxtBC`, `buffer`, τ_L, and $\vec{\tau}_{\texttt{state}}$, as described above. Initially, $\texttt{state} := \vec{\tau}_{\texttt{state}} := \texttt{NxtBC} := \varepsilon$, $\texttt{buffer} := \emptyset$, $\tau_L = 0$.
For each party $P_i \in \mathcal{P}$ the functionality maintains a pointer $\texttt{pt}_i$ (initially set to 1) and a current-state view $\texttt{state}_i := \varepsilon$ (initially set to empty). The functionality also keeps track of the timed honest-input sequence in a vector $\vec{\mathcal{I}}_H^T$ (initially $\vec{\mathcal{I}}_H^T := \varepsilon$).

Party management: The functionality maintains the set of registered parties $\mathcal{P}$, the (sub-)set of honest parties $\mathcal{H} \subseteq \mathcal{P}$, and the (sub-set) of de-synchronized honest parties $\mathcal{P}_{DS} \subset \mathcal{H}$ (following the definition in the previous paragraph). The sets $\mathcal{P}, \mathcal{H}, \mathcal{P}_{DS}$ are all initially set to $\emptyset$. When a new honest party is registered at the ledger, if it is registered with the clock already then it is added to the party sets $\mathcal{H}$ and $\mathcal{P}$ and the current time of registration is also recorded; if the current time is $\tau_L > 0$, it is also added to $\mathcal{P}_{DS}$. Similarly, when a party is deregistered, it is removed from both $\mathcal{P}$ (and therefore also from $\mathcal{P}_{DS}$ or $\mathcal{H}$). The ledger maintains the invariant that it is registered (as a functionality) to the clock whenever $\mathcal{H} \neq \emptyset$. A party is considered fully registered if it is registered with the ledger and the clock.

Upon receiving any input I from any party or from the adversary, send $(\text{CLOCK-READ}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and upon receiving response $(\text{CLOCK-READ}, \text{sid}_C, \tau)$ set $\tau_L := \tau$ and do the following:

1. Let $\widehat{\mathcal{P}} \subseteq \mathcal{P}_{DS}$ denote the set of desynchronized honest parties that have been registered (continuously) since time $\tau' < \tau_L - \texttt{Delay}$ (to both ledger and clock). Set $\mathcal{P}_{DS} := \mathcal{P}_{DS} \setminus \widehat{\mathcal{P}}$.

2. If I was received from an honest party $P_i \in \mathcal{P}$:

   (a) Set $\vec{\mathcal{I}}_H^T := \vec{\mathcal{I}}_H^T || (I, P_i, \tau_L)$;

   (b) Set $\vec{N} = (\vec{N}_1, \ldots, \vec{N}_\ell) := \mathsf{ExtendPolicy}(\vec{\mathcal{I}}_H^T, \texttt{state}, \texttt{NxtBC}, \texttt{buffer}, \vec{\tau}_{\texttt{state}})$ and if $\vec{N} \neq \varepsilon$ set $\texttt{state} := \texttt{state} || \mathsf{Blockify}(\vec{N}_1) || \ldots || \mathsf{Blockify}(\vec{N}_\ell)$ and $\vec{\tau}_{\texttt{state}} := \vec{\tau}_{\texttt{state}} || \tau_L^\ell$, where $\tau_L^\ell = \tau_L || \ldots, || \tau_L$.

   (c) For each $\texttt{BTX} \in \texttt{buffer}$: if $\mathsf{Validate}(\texttt{BTX}, \texttt{state}, \texttt{buffer}) = 0$ then delete BTX from buffer. Also, reset $\texttt{NxtBC} := \varepsilon$.

   (d) If there exists $P_j \in \mathcal{H} \setminus \mathcal{P}_{DS}$ such that $|\texttt{state}| - \texttt{pt}_j > \texttt{windowSize}$ or $\texttt{pt}_j < |\texttt{state}_j|$, then set $\texttt{pt}_k := |\texttt{state}|$ for all $P_k \in \mathcal{H} \setminus \mathcal{P}_{DS}$.

3. Depending on the input I and the ID of the sender, execute the respective code:

- *Submiting a transaction:*
  If $I = (\text{SUBMIT}, \text{sid}, \texttt{tx})$ and is received from a party $P_i \in \mathcal{P}$ or from $\mathcal{A}$ (on behalf of a corrupted party P_i) do the following
  (a) Choose a unique transaction ID txid and set $\texttt{BTX} := (\texttt{tx}, \text{txid}, \tau_L, P_i)$
  (b) If $\mathsf{Validate}(\texttt{BTX}, \texttt{state}, \texttt{buffer}) = 1$, then $\texttt{buffer} := \texttt{buffer} \cup \{\texttt{BTX}\}$.
  (c) Send $(\text{SUBMIT}, \texttt{BTX})$ to $\mathcal{A}$.
- *Reading the state:*
  If $I = (\text{READ}, \text{sid})$ is received from a fully registered party $P_i \in \mathcal{P}$ then set $\texttt{state}_i := \texttt{state}|_{\min\{\texttt{pt}_i, |\texttt{state}|\}}$ and return $(\text{READ}, \text{sid}, \texttt{state}_i)$ to the requestor. If the requestor is $\mathcal{A}$ then send $(\texttt{state}, \texttt{buffer}, \vec{\mathcal{I}}_H^T)$ to $\mathcal{A}$.
- *Maintaining the ledger state:*
  If $I = (\text{MAINTAIN-LEDGER}, \text{sid}, \text{minerID})$ is received by an honest party $P_i \in \mathcal{P}$ and (after updating $\vec{\mathcal{I}}_H^T$ as above) $\mathsf{predict\text{-}time}(\vec{\mathcal{I}}_H^T) = \widehat{\tau} > \tau_L$ then send $(\text{CLOCK-UPDATE}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$. Else send I to $\mathcal{A}$.
- *The adversary proposing the next block:*
  If $I = (\text{NEXT-BLOCK}, \text{hFlag}, (\text{txid}_1, \ldots, \text{txid}_\ell))$ is sent from the adversary, update $\texttt{NxtBC}$ as follows:
  (a) Set listOfTxid $\leftarrow \epsilon$
  (b) For $i = 1, \ldots, \ell$ do: if there exists $\texttt{BTX} := (x, \text{txid}, \text{minerID}, \tau_L, P_i) \in \texttt{buffer}$ with ID txid $=$ txid_i then set listOfTxid := listOfTxid$||\text{txid}_i$.
  (c) Finally, set $\texttt{NxtBC} := \texttt{NxtBC}||(\text{hFlag}, \text{listOfTxid})$ and output $(\text{NEXT-BLOCK}, ok)$ to $\mathcal{A}$.
- *The adversary setting state-slackness:*
  If $I = (\text{SET-SLACK}, (P_{i_1}, \widehat{\texttt{pt}}_{i_1}), \ldots, (P_{i_\ell}, \widehat{\texttt{pt}}_{i_\ell}))$, with $\{P_{i_1}, \ldots, P_{i_\ell}\} \subseteq \mathcal{H} \setminus \mathcal{P}_{DS}$ is received from the adversary $\mathcal{A}$ do the following:
  (a) If for all $j \in [\ell]$: $|\texttt{state}| - \widehat{\texttt{pt}}_{i_j} \leq \texttt{windowSize}$ and $\widehat{\texttt{pt}}_{i_j} \geq |\texttt{state}_{i_j}|$, set $\texttt{pt}_{i_1} := \widehat{\texttt{pt}}_{i_1}$ for every $j \in [\ell]$ and return $(\text{SET-SLACK}, ok)$ to $\mathcal{A}$.
  (b) Otherwise set $\texttt{pt}_{i_j} := |\texttt{state}|$ for all $j \in [\ell]$.
- *The adversary setting the state for desychronized parties:*
  If $I = (\text{DESYNC-STATE}, (P_{i_1}, \texttt{state}'_{i_1}), \ldots, (P_{i_\ell}, \texttt{state}'_{i_\ell}))$, with $\{P_{i_1}, \ldots, P_{i_\ell}\} \subseteq \mathcal{P}_{DS}$ is received from the adversary $\mathcal{A}$, set $\texttt{state}_{i_j} := \texttt{state}'_{i_j}$ for each $j \in [\ell]$ and return $(\text{DESYNC-STATE}, ok)$ to $\mathcal{A}$.

This concludes the description of the basic ledger functionality.

8.4 Bitcoin as a UC Protocol

8.4.1 Basics of Bitcoin

For the sake of self-containment, this section introduces the core algorithms of the Bitcoin protocol.

Notation

A *blockchain* $\mathcal{C} = \mathbf{B}_1, \ldots, \mathbf{B}_n$ is a (finite) sequence of blocks where each *block* $\mathbf{B}_i = \langle \mathtt{s}_i, \mathtt{st}_i, \mathtt{n}_i \rangle$ is a triple consisting of the *pointer* $\mathtt{s}_i$, the *state block* $\mathtt{st}_i$, and the *nonce* $\mathtt{n}_i$. The *head* of chain $\mathcal{C}$ is the block $\mathtt{head}(\mathcal{C}) := \mathbf{B}_n$ and the *length* $\mathtt{length}(\mathcal{C})$ of the chain is the number of blocks, i.e., $\mathtt{length}(\mathcal{C}) = n$. The chain $\mathcal{C}^{\lceil k}$ is the (potentially empty) sequence of the first $\mathtt{length}(\mathcal{C}) - k$ blocks of $\mathcal{C}$. A special block is the *genesis block* $\mathbf{G} = \langle \bot, \mathtt{gen}, \bot \rangle$ which contains the genesis state $\mathtt{gen} := \varepsilon$ and, as we will see later, is required to be the first block in the sequence.

The *state* $\vec{\mathtt{st}}$ *encoded* in $\mathcal{C}$ is defined as a sequence of the corresponding state blocks, i.e., $\vec{\mathtt{st}} := \mathtt{st}_1 || \ldots || \mathtt{st}_n$. In other words, one should think of the blockchain $\mathcal{C}$ as an encoding of its underlying state $\vec{\mathtt{st}}$; such an encoding might, e.g., organize $\mathcal{C}$ is an efficient searchable data structure as is the case in the Bitcoin protocol where a blockchain is a linked list implemented with hash-pointers. In the protocol, the blockchain is the data structure storing a sequence of entries, often referred to as transactions. Furthermore, as in [KZZ16], in order to capture blockchains with syntactically different state encoding, we use an algorithm $\mathsf{blockify}_{\mathcal{B}}$ to map a vector of transactions into a state block. Thus, each block $\mathtt{st} \in \vec{\mathtt{st}}$ (except the genesis state) of the state encoded in the blockchain has the form $\mathtt{st} = \mathsf{blockify}_{\mathcal{B}}(\vec{N})$ where $\vec{N}$ is a vector of transactions.

Validity and Longest Valid Chains

For a blockchain $\mathcal{C}$ to be considered a valid blockchain, it needs to satisfy certain conditions. Concretely, the validity of a blockchain $\mathcal{C} = \mathbf{B}_1, \ldots, \mathbf{B}_n$ where $\mathbf{B}_i = \langle \mathtt{s}_i, \mathtt{st}_i, \mathtt{n}_i \rangle$ depends on two aspects: *chain-level* validity, also referred to as syntactic validity, and a *state-level* validity also referred to as semantic validity.

Algorithm $\mathsf{validStruct}^{\mathtt{D}}_{\mathcal{B}}(\mathcal{C})$

res $\leftarrow$ true
if $(\mathtt{length}(\mathcal{C}) = 0)$ **or** $(H[\mathtt{head}(\mathcal{C})] \geq \mathtt{D})$ **then**
  res $\leftarrow$ false
else if $\mathtt{length}(\mathcal{C}) = 1$ **then**
  res $\leftarrow (\mathcal{C} = \mathbf{G})$
else ▷ In this case, the chain is non-trivial and the most recent block is a valid proof-of-work.
  $\mathcal{C}' \leftarrow \mathcal{C}$
  $\langle \mathtt{s}', \cdot, \cdot \rangle \leftarrow \mathtt{head}(\mathcal{C}')$
  repeat
    $\mathcal{C}' \leftarrow \mathcal{C}'^{\lceil 1}$ ▷ Chop off the head of $\mathcal{C}'$.
    $\mathbf{B} := \langle \mathtt{s}, \mathtt{st}, \mathtt{n} \rangle \leftarrow \mathtt{head}(\mathcal{C}')$
    if $(H[\mathbf{B}] \neq \mathtt{s}')$ **or** $(\mathtt{length}(\mathcal{C}') > 1$ **and** $H[\mathtt{head}(\mathcal{C})] \geq \mathtt{D})$ **or** $(\mathtt{length}(\mathcal{C}') = 1$ **and** $\mathbf{B} \neq \mathbf{G})$ **then**
      res $\leftarrow$ false
    else
      $\mathtt{s}' \leftarrow \mathtt{s}$
  until res = false **or** $\mathtt{length}(\mathcal{C}') = 1$
return res

Figure 8.6: The syntactic validity check.

Syntactic validity. This is defined with respect to a difficulty parameter $\mathtt{D} \in [2^{\kappa}]$, where κ is the security parameter, and a given hash function $H(\cdot) : \{0,1\}^* \to \{0,1\}^{\kappa}$; at its core, it requires that, for each $i > 1$, the value $\mathtt{s}_i$ contained in $\mathbf{B}_i$ satisfies $\mathtt{s}_i = H[\mathbf{B}_{i-1}]$ and that additionally $H[\mathbf{B}_i] < \mathtt{D}$ holds (for non-genesis blocks), where we interpret the output of the hash-function as an integer in this comparison. The algorithm is given in Figure 8.6. Note that for notational simplicity, we omit the hash-function as an explicit superscript.

Semantic validity. This is defined on the state $\vec{\mathtt{st}}$ encoded in the blockchain $\mathcal{C}$ and specifies whether this content is valid (which might depend on a particular application). Recall that the validation predicate Validate defined in the ledger functionality plays a similar role. For example, a natural and generic semantic validity of the blockchain can be defined algorithm that we denote $\mathsf{isvalidstate}_{\mathcal{B}}$ which builds upon a validation predicate for transactions, such as Validate. Recall that in the general ledger description, Validate might depend on some associated

metadata; although this might be useful to capture alternative blockchains, it is not the case for Bitcoin and to avoid confusion, throughout this section we use $\mathsf{ValidTx}_{\mathbb{B}}$ to refer to a generic validation predicate which ignores all information other than the state and the transaction that is being validated. The pseudo-code of the algorithm $\mathsf{isvalidstate}_{\mathbb{B}}$ which builds upon $\mathsf{ValidTx}_{\mathbb{B}}$ is provided below. In a nutshell, the algorithm checks that a given blockchain state can be built in an iterative manner, such that each contained transaction is considered valid according to $\mathsf{ValidTx}_{\mathbb{B}}$ upon insertion. It further ensures that the state starts with the genesis state and that state blocks contain a special *coin-base* transaction $\mathtt{tx}^{\text{coin-base}}_{\text{minerID}}$ which assigns them to a miner.

Algorithm $\mathsf{isvalidstate}_{\mathbb{B}}(\vec{\mathtt{st}})$

Let $\vec{\mathtt{st}} := \mathtt{st}_1 || \ldots || \mathtt{st}_n$
for each $\mathtt{st}_i$ **do**
  Extract the transaction sequence $\vec{\mathtt{tx}}_i \leftarrow \mathtt{tx}_{i,1}, \ldots, \mathtt{tx}_{i,n_i}$ contained in $\mathtt{st}_i$
$\vec{\mathtt{st}}' \leftarrow \mathtt{gen}$ ▷ Initialize the genesis state
for $i = 1$ to n **do**
  if the first transaction in $\vec{\mathtt{tx}}_i$ is not a coin-base transaction **return** false
  $\vec{N}_i \leftarrow \mathtt{tx}_{i,1}$
  for $j = 2$ to $|\vec{\mathtt{tx}}_i|$ **do**
    $\mathtt{st} \leftarrow \mathsf{blockify}_{\mathbb{B}}(\vec{N}_i)$
    if $\mathsf{ValidTx}_{\mathbb{B}}(\mathtt{tx}_{i,j}, \vec{\mathtt{st}}' || \mathtt{st}) = 0$ **return** false
    $\vec{N}_i \leftarrow \vec{N}_i || \mathtt{tx}_{i,j}$
  $\vec{\mathtt{st}}' \leftarrow \vec{\mathtt{st}}' || \mathtt{st}_i$
return true

Definition 8.4.1. A chain $\mathcal{C}$ is valid if it satisfies syntactic and semantic validity, i.e., if, for the chain and its encoded state $\vec{\mathtt{st}}$, the predicate

$$\mathsf{isvalidchain}^{\mathtt{D}}_{\mathbb{B}}(\mathcal{C}) := \mathsf{validStruct}^{\mathtt{D}}_{\mathbb{B}}(\mathcal{C}) \wedge \mathsf{isvalidstate}_{\mathbb{B}}(\vec{\mathtt{st}})$$

evaluates to true.

Longest valid chain. In the Bitcoin protocol, the notion of the *longest valid chain* is very crucial. The reason is that the party defines the ledger state at a certain time as a prefix of the state encoded in the longest valid chain it knows at that time. We stick to the nomenclature of [GKL15] and call the function $\mathsf{maxvalid}_{\mathbb{B}}(\mathcal{C}_1, \ldots, \mathcal{C}_k)$.

Algorithm $\mathsf{maxvalid}^{\mathsf{D}}_{\mathbb{B}}(\mathcal{C}_1, \ldots, \mathcal{C}_k)$

$\mathcal{C}_{temp} \leftarrow \varepsilon$
for $i = 1$ to k **do**
  if $\mathsf{isvalidchain}^{\mathsf{D}}_{\mathbb{B}}(\mathcal{C}_i)$ **and** $(\mathtt{length}(\mathcal{C}_i) > \mathtt{length}(\mathcal{C}_{temp}))$ **then**
    $\mathcal{C}_{temp} \leftarrow \mathcal{C}_i$
return $\mathcal{C}_{temp}$

Extending Chains and Proofs-of-Work

A core step in Bitcoin is to extend a given chain $\mathcal{C}$ by a new block $\mathbf{B}$ (with certain state content) to yield a longer chain $\mathcal{C}||\mathbf{B}$. As presented in [GKL15] this can be captured by an algorithm $\mathsf{extendchain}_{\mathsf{D}}(\cdot)$ that takes a chain $\mathcal{C}$, a state block $\mathtt{st}$ and the number of attempts q as inputs. It tries to find a proof-of-work which allows to extend the $\mathcal{C}$ by a block which encodes $\mathtt{st}$.

Algorithm $\mathsf{extendchain}_{\mathsf{D}}(\mathcal{C}, \mathtt{st}, q)$

Input: Chain $\mathcal{C}$ is valid with state $\vec{\mathtt{st}}$. The state $\vec{\mathtt{st}}||\mathtt{st}$ is valid.
  Set $\mathbf{B} \leftarrow \perp$
  $\mathtt{s} \leftarrow H[\mathtt{head}(\mathcal{C})]$ ▷ Compute the pointer $\mathtt{s}$ of the new block
  for $i \in \{1, \ldots, q\}$ **do**
    Choose nonce $\mathtt{n}$ uniformly at random from $\{0,1\}^{\kappa}$ and set $\mathbf{B} \leftarrow \langle \mathtt{s}, \mathtt{st}, \mathtt{n} \rangle$.
    if $H[\mathbf{B}] < \mathtt{D}$ **then**
      break
  if $\mathbf{B} \neq \perp$ **then**
    $\mathcal{C} \leftarrow \mathcal{C}||\mathbf{B}$
  return $\mathcal{C}$

8.4.2 Overview and Modeling Decisions

In Bitcoin, each party maintains a local blockchain which initially consists of the genesis block. The chains of honest parties might differ (but as we will prove, it will have a common prefix which will define the ledger state). New transactions are added in a 'mining process'. First, a party collects valid transactions (according to $\mathsf{ValidTx}_{\mathbb{B}}$) and creates a new state block $\mathtt{st}$ using $\mathsf{blockify}_{\mathbb{B}}$. Next, the party attempts to mine a new block by solving a puzzle (and hence finding a proof-of-work) which upon success could then be validly added to their local blockchain. After each mining

attempt parties will multicast their current chain. A party will replace its local chain if it obtains or receives a longer valid chain. When queried to output the state of the ledger, a party reports a prefix of the state encoded in its longest valid chain — obtained by ignoring (or chopping-off) the most recent T blocks (a party outputs ε if the state has less than T blocks). This behavior will ensure that all honest parties output a consistent ledger state. T is a crucial parameter of the Bitcoin protocol and typically, the guarantees of the security statements depend on T (and in addition on the usual security parameter κ).

The Round Structure

As already mentioned in the introduction, we model Bitcoin as a semi-synchronous protocol: The protocol can proceed in rounds — enabled by having access to a global synchronization clock $\mathcal{G}_{\text{CLOCK}}$— but is not aware of the actual delay of the network. In each round, two logical tasks have to be executed: an *updating* or information-fetching step (where new messages from the network are processed) and a *working* or mining-step, where each party tries to extend its local chain.

To simplify the UC activation handling in the analysis, we divide each logical round into two sub-rounds (where each sub-round corresponds to a logical task; see below for more details). This means that each logical round correspond to two actual clock-ticks (also known as mini-rounds in the MPC literature). We say that a protocol is in round r if the current time of the clock is $\tau \in \{2r, 2r+1\}$.

Having two clock-ticks per round is a standard way to model in synchronous UC that messages (e.g., a block) sent within a round are delivered at the beginning of the next round. In our case, each round is divided into two mini-rounds, where each mini-round corresponds to a clock tick. We treat the first mini-round as the *updating mini-round* (fetch messages from the network to obtain messages sent previous rounds) and the second mini-round as the *working mini-round* (solving the puzzle and multicasting solutions).

Handling Interrupts

A protocol command might consists of a sequence of operations. However, certain operations, such as sending a message to another party, result in

the protocol machine losing the activation token. We briefly describe a standard way to formalize that a party that loses an activation in the middle of a multi-step command is able to resume and complete the command following the implicit proposal of [KMTZ13]. Their mechanism can be made explicit by introducing an anchor a that stores a pointer to the current operation; the protocol associates each anchor with such a multiple command and an input I, so that when such an input is received it directly jumps to the stored anchor, executes the next operation(s) and updates (increases) the anchor before releasing the activation. We refer to such an execution as being *I-interruptible*.

As an example, consider a protocol that requires that upon receiving input I, the party should run a command that consists of m steps Step 1, Step 2, ..., Step m, but some of these steps might result in the party losing its activation. Running this command in an *I-interruptible* manner means executing the following code: Upon receiving input I if $a < m$ go to Step a and increase $a = a + 1$ before executing the first operation that releases the activation; otherwise go to Step 1 and set $a = 2$ before executing any operation that releases the activation.

8.4.3 The Formal Protocol Description

We can now formally define our blockchain protocol $\mathsf{Ledger\text{-}Protocol}_{q,\mathtt{D},T}$ (we usually omit the parameters when clear from the context). The protocol allows an arbitrary number of parties/miners to communicate by means of a multicast network $\mathcal{F}_{\text{N-MC}}^{\Delta}$. Note that this means that the adversary can send different messages to different parties. New miners might dynamically join or leave the protocol by means of the registration/de-registration commands: when they join they register with all associated functionalities and when they leave they deregister.[18] The pseudo-code of this UC blockchain protocol is given in the remainder of this section. The general structure of our UC blockchain protocol $\mathsf{Ledger\text{-}Protocol}_{q,\mathtt{D},T}(P)$ is given below.

The Bitcoin ledger protocol assumes as hybrids a random oracle $\mathcal{F}_{\text{RO}}$, a network $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ for blockchains, a network $\mathcal{F}_{\text{N-MC}}^{\mathsf{tx}}$ for transcations, and clock $\mathcal{G}_{\text{CLOCK}}$. Note that the two networks are simply (named) instances of

[18] Note that when a party registers to a local functionality such as the network or the random oracle it does not lose its activation token. This is a subtle point to ensure that the real and ideal worlds are in-sync regarding activations.

$\mathcal{F}_{\text{N-MC}}^{\Delta}$ and can be realized from a single network $\mathcal{F}_{\text{N-MC}}^{\Delta}$ using different message-IDs. The protocol is parametrized by q, $\mathtt{D}$, T where q is the number of mining attempts per round, $\mathtt{D}$ is the difficulty of the proof-of-work, and T is the number of blocks chopped off to obtain the ledger state.

We provide the description below and discuss each part in more detail in the following paragraph.

Protocol Ledger-Protocol$_{q,\mathtt{D},T}(P)$

Variables and Initial Values:

- The protocol stores a local (working) chain $\mathcal{C}_{loc}$ which initially contains the genesis block, i.e., $\mathcal{C}_{loc} \leftarrow (\mathbf{G})$.
- It additionally manages a separate chain $\mathcal{C}_{\text{exp}}$ to store the current chain whose encoded state $\vec{\mathtt{st}}$ is exported as the ledger state (initially this chain contains the genesis block).
- Variable `isInit` stores the initialization status. Initially this variable is false.
- `buffer` contains the list of transactions obtained from the network. Initially, this buffer is empty.
- A time stamp t to remember when this party was last active (initially, $t = 0$) and a flag WELCOME to indicate whether a indication was received that a new party joined the network (initially WELCOME $= 0$).
- The party stores its registration status to the hybrid functionalities internally. We do not introduce an explicit name for this variable.

Registration/De-Registration:

- Upon receiving (REGISTER, sid) do the following: if this party is registered with the clock, then send (REGISTER, sid) to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$, $\mathcal{F}_{\text{N-MC}}^{\mathsf{tx}}$, and $\mathcal{F}_{\text{RO}}$ and output (REGISTER, sid, P); otherwise, ignore the input.
- Upon receiving (DE-REGISTER, sid), send (DE-REGISTER, sid) to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$, $\mathcal{F}_{\text{N-MC}}^{\mathsf{tx}}$, and $\mathcal{F}_{\text{RO}}$. Set all variables back to their initial values and return (DE-REGISTER, sid, P).
- Upon receiving (IS-REGISTERED, sid), return (REGISTER, sid, 1) if this party is registered with the network and the random oracle. Otherwise, return (REGISTER, sid, 0).
- Upon receiving (REGISTER, sid$_C$) (for the global clock), send (REGISTER, sid$_C$) to $\mathcal{G}_{\text{CLOCK}}$ and return whatever $\mathcal{G}_{\text{CLOCK}}$ returns.
- Upon receiving (DE-REGISTER, sid$_C$) (for the global clock), send (DE-REGISTER, sid$_C$) to $\mathcal{G}_{\text{CLOCK}}$ and return whatever $\mathcal{G}_{\text{CLOCK}}$ returns.

Ledger-Queries:
Ledger queries are only answered once registered.

- Upon receiving (SUBMIT, sid, tx), set buffer ← buffer||tx, and send (MULTICAST, sid, tx) to $\mathcal{F}_{\text{N-MC}}^{\text{tx}}$.
- Upon receiving (READ, sid) send (CLOCK-READ, sid_C) to $\mathcal{G}_{\text{CLOCK}}$, receive as answer (CLOCK-READ, sid_C, τ) and proceed as follows:
  if τ corresponds to an update mini-round and $t < \tau$ and isInit **then**
  Execute sub-protocol **FetchInformation** and set $t \leftarrow \tau$.
  Let $\vec{\text{st}}$ be the encoded state in $\mathcal{C}_{\text{exp}}$
  Return (READ, sid, $\vec{\text{st}}^{\lceil T}$).
- Upon receiving (MAINTAIN-LEDGER, sid, minerID) execute in a (MAINTAIN-LEDGER, sid, minerID)-interruptible manner the following:
  1. If isInit = false, then set all variables to their initial values, set isInit ← true and output (MULTICAST, sid, NEW-PARTY) to $\mathcal{F}_{\text{N-MC}}^{\text{tx}}$.
  2. Execute sub-protocol **Ledger-Maintenance**

Handling other external calls:

- Upon receiving (CLOCK-READ, sid_C) forward the query to $\mathcal{G}_{\text{CLOCK}}$ and return whatever is received as answer from $\mathcal{G}_{\text{CLOCK}}$
- Upon receiving (CLOCK-UPDATE, sid_C), remember that a clock-update was received in the current mini-round for later reference. If this protocol instance is currently only registered to the clock (and no other functionality), then forward (CLOCK-UPDATE, sid_C) to $\mathcal{G}_{\text{CLOCK}}$.

Registration, De-Registration and Initialization

The registration process in the protocol works as follows. If a party receives (REGISTER, sid) from the environment it registers at the random oracle and the network. Since the clock is a shared functionality, the registrations are fully controlled by the environment and thus the protocol relays such registration queries to the clock. Only if a party is registered to the clock already, it reacts to such REGISTER queries and otherwise stays idle. Once registration has succeeded the party returns activation to the environment. Upon the next activation to maintain the ledger (MAINTAIN-LEDGER), the party initializes its local variables, multicasts a special NEW-PARTY message over the network, and executes the main maintenance sub-protocol (in an interruptible manner as further explained below).

De-registering from the ledger (via a query (DE-REGISTER, sid)) from the environment) works analogously, upon which the party erases all its state and becomes idle until its is freshly invoked with a REGISTER-query.

Recall that the notion of de-synchronized parties is strongly connected to its registration: if an active honest party is not registered with the clock or not registered to all hybrids for long enough after joining the protocol execution at some time $\tau > 0$, it is considered de-synchronized (and otherwise the party is synchronized). In particular, honest parties that register at the onset of the protocol execution are synchronized (until they get corrupted or de-registered from the clock).

Ledger-Specific Queries

Ledger specific queries are the specific features that one wishes to implement. Our very basic ledger supports three operations (after registration):

Submitting a transaction. This one is very simple: when given a transaction a party multicasts the transaction.

Ledger maintenance. Ledger maintenance refers to activating the main mining procedure of Bitcoin and is given in Figure 8.7. Since ledger maintenance consists of several complex steps that in particular lose activations, the execution proceeds in an interruptible manner as explained in Section 8.4.2. The main structure of maintenance enforces the mini-round structure: in a working mini-round, the protocol tries to obtain the solution to a proof-of-work puzzle for a newly generated state block. The core sub-protocol thereby is:

Sub-Protocol ExtendState(st)

$\mathcal{C}_{\text{new}} \leftarrow \mathsf{extendchain}_{\mathtt{D}}(\mathcal{C}_{loc}, \mathtt{st}, q)$
if $\mathcal{C}_{\text{new}} \neq \mathcal{C}_{loc}$ **then**
  Update the local chain, i.e., $\mathcal{C}_{loc} \leftarrow \mathcal{C}_{\text{new}}$.
Send $(\text{MULTICAST}, \text{sid}, \mathcal{C}_{loc})$ to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ ▷ Multicast current chain

It then enters an idle mode for maintenance queries until the clock advances and enters an update mini-round where new information is fetched from the network.

Sub-Protocol FetchInformation

Send (FETCH, sid) to $\mathcal{F}_{\text{N-MC}}^{\mathtt{bc}}$; denote the response from $\mathcal{F}_{\text{N-MC}}^{\mathtt{bc}}$ by (FETCH, sid, b).
Extract chains $\mathcal{C}_1, \ldots, \mathcal{C}_k$ from b.
$\mathcal{C}_{loc}, \mathcal{C}_{\text{exp}} \leftarrow \mathsf{maxvalid}_{\mathtt{p}}^{\mathtt{p}}(\mathcal{C}_{loc}, \mathcal{C}_{\text{exp}}, \mathcal{C}_1, \ldots, \mathcal{C}_k)$
Send (FETCH, sid) to $\mathcal{F}_{\text{N-MC}}^{\mathtt{tx}}$; denote the response from $\mathcal{F}_{\text{N-MC}}^{\mathtt{tx}}$ by (FETCH, sid, b).
Extract received transactions $(\mathtt{tx}_1, \ldots, \mathtt{tx}_k)$ from b.
Set $\mathtt{buffer} \leftarrow \mathtt{buffer} || (\mathtt{tx}_1, \ldots, \mathtt{tx}_k)$.
If a NEW-PARTY message was received, set WELCOME $\leftarrow 1$. Otherwise, set WELCOME $\leftarrow 0$.
Remove all transactions from `buffer` which are invalid with respect to $\vec{\mathtt{st}}^{\lceil T}$

Again the protocol is idle for maintenance queries until the clock advances.

Reading the state. When asked to report the current ledger state, the protocol outputs the prefix of the exported state, i.e., a prefix of the state encoded in $\mathcal{C}_{\text{exp}}$. By the mini-round structure, the exported state is updated exactly once in every update mini-rounds (after initialization is complete).

Predictable Synchronization Pattern

We now show that the ledger protocol has a predictable synchronization pattern according to Definition 8.2.1.

Lemma 8.4.2. *The protocol* $\mathsf{Ledger\text{-}Protocol}_{q,\mathcal{D},T}$ *satisfies Definition 8.2.1. More specifically, there is a predicate* $\mathsf{predict\text{-}time}_{BC}$ *that predicts the synchronization pattern of the UC Bitcoin protocol as required by Definition 8.2.1.*

Proof Sketch. This is straightforward to see for our ledger protocol (and all protocols that share the same structure) in all the respective hybrid worlds they are executed. The predicate predict-time can be implemented as follows: browse through the entire sequence $\vec{\mathcal{I}}_H^T$ and determine how many times the clock advances. The clock advances for the first time, when all miners got sufficient maintain commands to complete their mini-round operation, followed by a clock-update command. By definition of Ledger-Protocol, this implies that each party has sent a clock-update to the clock and hence the clock advances. By an inductive argument, whenever the clock has ticked, the check when the clock advances the next time is

Sub-Protocol Ledger-Maintenance

This sub-protocol is executed in a (MAINTAIN-LEDGER, *sid*, *minerID*)*-interruptible manner*

Step 1: If a $(\text{CLOCK-UPDATE}, \text{sid}_C)$ has been received during this update mini-round then send $(\text{CLOCK-UPDATE}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ (if it hasn't been sent already in the current mini-round), and in the next activation go to the next step. Else in the next activation repeat this step.

Step 2: Send $(\text{CLOCK-READ}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$, receive as answer $(\text{CLOCK-READ}, \text{sid}_C, \tau)$, and proceed as follows.

- **if** τ corresponds to a working mini-round **then**
  - ▷ Generate a new block: extract transactions and form a state-block and append
  - Let $\vec{\mathtt{st}}$ be the encoded state in $\mathcal{C}_{loc}$
  - Set $\mathtt{buffer}' \leftarrow \mathtt{buffer}$
  - Parse $\mathtt{buffer}'$ as sequence $(\mathtt{tx}_1, \ldots, \mathtt{tx}_n)$
  - Set $\vec{N} \leftarrow \mathtt{tx}^{\text{coin-base}}_{\text{minerID}}$
  - Set $\mathtt{st} \leftarrow \mathsf{blockify}_{\mathbb{B}}(\vec{N})$
  - **repeat**
    - Let $(\mathtt{tx}_1, \ldots, \mathtt{tx}_n)$ be the current list of (remaining) transactions in $\mathtt{buffer}'$
    - **for** $i = 1$ to n **do**
      - **if** $\mathsf{ValidTx}_{\mathbb{B}}(\mathtt{tx}_i, \vec{\mathtt{st}}||\mathtt{st}) = 1$ **then**
        - $\vec{N} \leftarrow \vec{N}||\mathtt{tx}_i$
        - Remove $\mathtt{tx}$ from $\mathtt{buffer}'$
        - Set $\mathtt{st} \leftarrow \mathsf{blockify}_{\mathbb{B}}(\vec{N})$
  - **until** $\vec{N}$ does not increase anymore
  - Execute **ExtendState(st)**
  - If the flag WELCOME $= 1$, send $(\text{MULTICAST}, \text{sid}, \mathtt{tx})$ to $\mathcal{F}^{\mathtt{tx}}_{\text{N-MC}}$ for all $\mathtt{tx} \in \mathtt{buffer}$.
  - Go to step 3 in the next activation.
- **else**
  - Go to the beginning of step 2 in the next activation.

Step 3:
If a $(\text{CLOCK-UPDATE}, \text{sid}_C)$ has been received during this working round then send $(\text{CLOCK-UPDATE}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$, and in the next activation go to the next step. Else in the next activation repeat this step.

Step 4:
Send $(\text{CLOCK-READ}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$, receive as answer $(\text{CLOCK-READ}, \text{sid}_C, \tau)$, and proceed as follows.

- **if** τ corresponds to an update mini-round **then**
  - If $t < \tau$ execute **FetchInformation** and set $t \leftarrow \tau$.
  - Go to step 1 in the next activation.
- **else**
  - Go to the beginning of step 4 in the next activation.

Figure 8.7: The maintenance procedure of the UC Bitcoin protocol.

checked exactly the same way. Overall, this allows to check whether the next activation of an honest party, given the history of activations will provoke a clock update. Note that only an activation of an honest party can make the clock advance. □

8.5 The Bitcoin Ledger

We next show how to instantiate the ledger functionality from Section 8.3 with appropriate parameters so that it is implemented by protocol Ledger-Protocol. The proof of this appears in the next section. To define this Bitcoin ledger $\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$, we give the specific instantiations of the relevant functions Validate, Blockify, ExtendPolicy, and predict-time.

Synchrony pattern. First, predict-time is defined to be $\mathsf{predict\text{-}time}_{BC}$ to reflect the synchronization pattern of the UC Bitcoin protocol as described in the proof of Lemma 8.4.2. This shows the dependency of the realized ledger from the protocol that achieves it.

State-buffer-validation. Similarly, in case of Validate we use the same predicate as the protocol uses to validate the states: For a given transaction `tx` and a given ledger-state `state`, the predicate decides whether this transaction is valid with respect to `state`. Given such a validation predicate, the ledger validation predicate takes a specific simple form which, excludes dependency on anything other than the transaction `tx` and `state`, i.e., for any values of txid, τ_L, P_i, and `buffer`:

$$\mathsf{Validate}((\mathtt{tx}, \text{txid}, \tau_L, P_i), \mathtt{state}, \mathtt{buffer}) := \mathsf{ValidTx}_{\mathbb{B}}(\mathtt{tx}, \mathtt{state}).$$

Ledger-output format. As with the above parameters, the function Blockify is defined to be $\mathsf{blockify}_{\mathbb{B}}$, i.e., the function used in the UC Bitcoin protocol. In principle, any formatting function can be used and the security proof goes through (as long as the same function is used in the protocol Ledger-Protocol and functionality $\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$). However, as we observe below in Definition 8.5.1, a meaningful Blockify should be in certain relation with the ledger's Validate predicate. This relation is satisfied by the Bitcoin protocol.

The ledger policy. Finally, we define ExtendPolicy. At a high level, upon receiving a list of possible candidate blocks which should go into the state of the ledger, ExtendPolicy does the following: for each block it first verifies that the blocks are valid with respect to the state they extend. Only valid blocks might be added to the state. Moreover, ExtendPolicy ensures the following property:

1. The speed of the ledger is not too slow. This is implemented by defining an upper bound $\texttt{maxTime}_{\texttt{window}}$ on the time interval (number of clock-ticks) within which at least `windowSize` state blocks have to be added. This is known as minimal chain-growth.

2. The speed of the ledger is not too fast. This is implemented by defining a lower bound $\texttt{minTime}_{\texttt{window}}$ on the time interval (number of clock-ticks), such that the adversary is not allowed to propose new blocks if `windowSize` or more blocks have already been added during that time interval.

3. The adversary cannot create too many blocks with arbitrary (but valid) contents. This is formally enforced by defining an upper bound η on the number of these so-called adversarial blocks within a sequence of `windowSize` state blocks. This is known as chain quality. Formally, this is enforced by requiring that a certain fraction of blocks need to satisfy higher quality standards (to model blocks that are honestly generated).

4. Last but not least, ExtendPolicy guarantees that if a transaction is "old enough", and still valid with respect to the actual state, then it is included into the state. This is a weak form of guaranteeing that a transaction will make it into the state unless it is in conflict. As we show in Section 8.8, this guarantee can be amplified by using digital signatures.

In order to enforce these policies, ExtendPolicy first defines alternative blocks which satisfy all of the above criteria in an ideal way, and whenever it catches the adversary in trying to propose blocks that do not obey the policies, it punishes the adversary by proposing its own generated blocks. In particular, if the adversary violates the policy regarding minimal chain-growth, the ExtendPolicy will directly propose a sequence of complying

blocks. The precise formal description of the extend policy (as pseudocode) for $\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$ is given in Appendix B.2 for completeness.

On the relation between Blockify and Validate. As already discussed above, ExtendPolicy guarantees that the adversary cannot block the extension of the state indefinitely, and that occasionally an honest miner will create a block. These are implications of the chain-growth and chain-quality properties from [GKL15]. However, our generic ExtendPolicy makes explicit that a priori, we cannot exclude that the chain always extends with blocks that include, for example, only a coin-base transaction, i.e., any submitted transaction is ignored and never inserted into a new block. This issue is an orthogonal one to ensuring that honest transactions are not invalidated by adversarial interaction—which, as argued in [GKL15], is achieved by adding digital signatures.

To see where this could be problematic in general, consider a blockify that, at a certain point, creates a block that renders all possible future transactions invalid. Observe that this does not mean that our protocol is insecure and that this is as well possible for the protocols of [GKL15, PSS17]; indeed our proof shows that the protocol will give exactly the same guarantees as an $\mathcal{G}_{\text{LEDGER}}$ parametrized with such an algorithm Blockify.

Nonetheless, a look in reality indicates that this situation never occurs with Bitcoin. To capture that this is the case, Validate and Blockify need to be in a certain relation with each other. Informally, this relation should ensure that the above sketched situation does not occur, i.e., Blockify should "not affect" the "true validity" of a transaction. A way to ensure this, which is already implemented by the Bitcoin protocol, is by restricting Blockify to only make an invertible manipulation of the blocks when they are inserted into the state—e.g., be an encoding function—and define Validate to depend on the inverse of Blockify. This is captured in the following definition.

Definition 8.5.1. A co-design of Blockify and Validate is called *non-self-disqualifying* if there exists an efficiently computable function Dec mapping outputs of Blockify to vectors $\vec{N}$ such that there exists a validate predicate Validate$'$ for which the following two properties hold for any possible state $\texttt{state} = \texttt{st}_1||\ldots||\texttt{st}_\ell$, buffer $\texttt{buffer}$ vectors $\vec{N} := (\texttt{tx}_1, \ldots, \texttt{tx}_m)$, and

transaction $\mathtt{tx}$:

1.) $\mathsf{Validate}(\mathtt{tx}, \mathtt{state}, \mathtt{buffer})$
$= \mathsf{Validate}'(\mathtt{tx}, \mathsf{Dec}(\mathtt{st}_1)|| \ldots ||\mathsf{Dec}(\mathtt{st}_\ell), \mathtt{buffer})$

2.) $\mathsf{Validate}(\mathtt{tx}, \mathtt{state}||\mathsf{Blockify}(\vec{N}), \mathtt{buffer})$
$= \mathsf{Validate}'(\mathtt{tx}, \mathsf{Dec}(\mathtt{st}_1)|| \ldots ||\mathsf{Dec}(\mathtt{st}_\ell)||\vec{N}, \mathtt{buffer})$

We remark that the actual validation of Bitcoin does satisfy the above definition, since a transaction is only rendered invalid with respect to the state if the coins it is trying to spend have already been spent, and this only depends on the transactions in the state and not the metadata added by $\mathsf{Blockify}$. Hence, in the following, we assume that $\mathsf{ValidTx}_{\mathbb{B}}$ and $\mathsf{blockify}_{\mathbb{B}}$ satisfy the relation in Definition 8.5.1.

8.6 Security Analysis

8.6.1 Overview

In this section we prove our main theorem, namely that, under appropriate assumptions, Bitcoin realizes the instantiation of the ledger functionality from the previous section. We prove our main theorem which can be described informally as follows:

Theorem (Informal). *For the security parameter κ and assuming that $\mathtt{windowSize} = \omega(\log \kappa)$, then the protocol* Ledger-Protocol *securely realizes the concrete ledger functionality $\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$ defined in the previous section. The assumptions on network delays and mining power, where mining power is roughly understood as the ability to find proofs of work via queries to the random oracle (and will be formally defined later) are as follows:*

- *In any round of the protocol execution, the collective mining power of the adversary, contributed by corrupted and temporarily de-synchronized miners, does not exceed the mining power of honest (and synchronized) parties. The exact relation additionally captures the (negative) impact of network delays on the coordination of mining power of honest parties.*

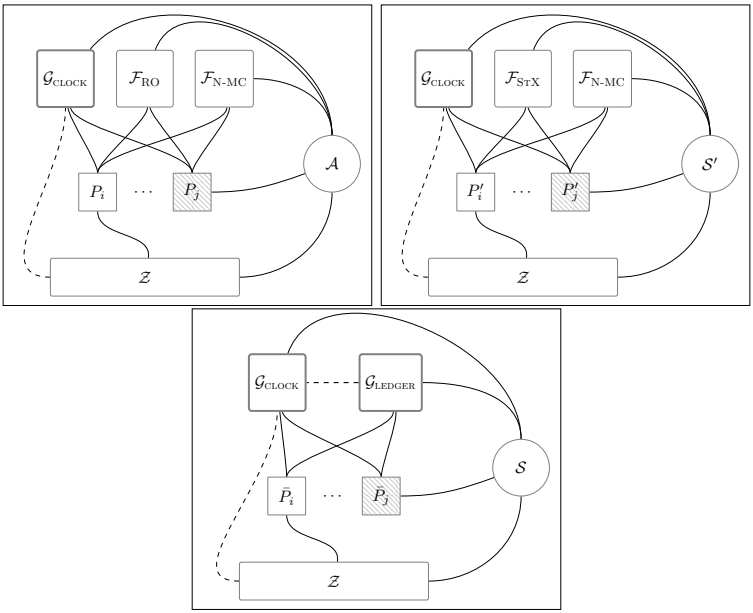


Figure 8.8: Illustration of the modular analysis.

- *No message can be delayed in the network by more than* $\Delta = O(1)$ *rounds.*

We prove the above theorem via what we believe is a useful modularization of the Bitcoin protocol (cf. Figure 8.8). Informally, this modularization distills out from the protocol a reactive *state-extend* subprocess which captures the lottery that decides which miner gets to advance the blockchain next and additionally the process of propagating this state to other miners. In Lemma 8.6.2 we show that the state-extend-and-exchange module/subprocess implements an appropriate reactive UC functionality $\mathcal{F}_{\text{StX}}$. We can then use the UC composition theorem which allows us to argue security of Ledger-Protocol in a simpler hybrid world where, instead of using this subprocess, parties make calls to the functionality $\mathcal{F}_{\text{StX}}$.

For the sake of generality, our treatment will assume a shared (global) clock functionality and therefore, our main proof follows the EUC-realization (externalized UC) notion introduced in [CDPW07b] which then implies

full GUC security as stated in [CDPW07b].

8.6.2 First Proof Step

In a first step, we distill out from the protocol Ledger-Protocol a state-extend module/subprocess, denoted as StateExchange-Protocol, and show, using a "game-hopping" argument, that a modular description of the Ledger-Protocol in which every party makes invocations of this subprocess, yields an equivalent protocol. We abstract the service provided by this sub-process by a new lottery-functionality denoted $\mathcal{F}_{\text{STX}}$. The modularized protocol, defined for the $\mathcal{F}_{\text{STX}}$-hybrid world is denoted by Modular-Ledger-Protocol.

As we prove, the sub-process StateExchange-Protocol UC-realizes $\mathcal{F}_{\text{STX}}$ and hence the original protocol Ledger-Protocol and the modularized protocol Modular-Ledger-Protocol are in fact indistinguishable. This final step is a direct consequence of the universal composition theorem: Ledger-Protocol UC emulates Modular-Ledger-Protocol where invocations of StateExchange-Protocol are replaced by invocations of $\mathcal{F}_{\text{STX}}$ (for appropriate parameters as precisely defined below).

Looking ahead, in the next section, we can hence focus on analyzing the simpler protocol Modular-Ledger-Protocol in order to show that the UC Bitcoin protocol realizes the Bitcoin Ledger of Section 8.5 — again by invoking the composition theorem.

The State-Exchange Functionality

The state-exchange functionality $\mathcal{F}_{\text{STX}}^{\Delta,p_H,p_A}$ allows parties to submit ledger states which are accepted with a certain probability. Accepted states are then multicast to all parties. Informally, it can be seen as as lottery on which (valid) states are exchanged among the participants. Note that for simplicity of notation we do not write the parameters when clear from the context.

Parties can use $\mathcal{F}_{\text{STX}}$ to multicast a valid state, but instead of accepting any submitted state and sending it to all (registered) parties, $\mathcal{F}_{\text{STX}}$ keeps track of all states that it ever saw, and implements the following mechanism upon submission of a state $\vec{\mathtt{st}}$ and a new block $\mathtt{st}$ from any party: If $\vec{\mathtt{st}}$ was previously accepted by $\mathcal{F}_{\text{STX}}$ and $\vec{\mathtt{st}}||\mathtt{st}$ is a valid new state, then $\mathcal{F}_{\text{STX}}$ accepts $\vec{\mathtt{st}}||\mathtt{st}$ with probability p_H (resp. p_A for dishonest

parties) and sends it to registered parties. Each submission is evaluated independently. The formal specification is found in Figure 8.9.

Realizing the State-Exchange Functionality

The state-exchange functionality is realized by the protocol given below. It is obtained by identifying the relevant instructions from the UC-ledger protocol. More precisely, protocol StateExchange-Protocol UC-realizes the $\mathcal{F}_{\text{StX}}$ functionality in the ($\mathcal{F}_{\text{RO}}$, $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$)-hybrid world. Note that $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ is a (named) instance of the $\mathcal{F}_{\text{N-MC}}^{\Delta}$ functionality. The protocol is parametrized by q and $\mathtt{D}$ where q is the number of mining attempts per submission attempt and $\mathtt{D}$ is the difficulty of the proof-of-work.

Protocol StateExchange-Protocol$_{q,\mathtt{D}}(P)$

Initialization:

The protocol maintains a tree $\mathcal{T}$ of all valid chains. Initially it contains the genesis chain ($\mathbf{G}$).

Registration/De-Registration:

- Upon receiving (REGISTER, sid) do the following: send (REGISTER, sid) to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ and $\mathcal{F}_{\text{RO}}$ and output (REGISTER, sid, P).
- Upon receiving (DE-REGISTER, sid), send (DE-REGISTER, sid) to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ and $\mathcal{F}_{\text{RO}}$. Set all variables back to their initial values and return (DE-REGISTER, sid, P).
- Upon receiving (IS-REGISTERED, sid), return (REGISTER, sid, 1) if this party is registered with $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ and $\mathcal{F}_{\text{RO}}$. Otherwise, return (REGISTER, sid, 0).

Exchange: *Exchange queries are only answered once registered.*

- Upon receiving (SUBMIT-NEW, sid, $\vec{\mathtt{st}}$, $\mathtt{st}$) do
  if $\mathsf{isvalidstate}_{\mathtt{D}}(\vec{\mathtt{st}}||\mathtt{st}) = 1$ **then** ▷ Check if there exists a chain in $\mathcal{T}$ which contains the state $\vec{\mathtt{st}}$
    if there exists $\mathcal{C} \in \mathcal{T}$ with $\vec{\mathtt{st}}$ **then**
      $\mathcal{C}_{\text{new}} \leftarrow \mathsf{extendchain}_{\mathtt{D}}(\mathcal{C}, \mathtt{st}, q)$ ▷ Try to extend the chain
      if $\mathcal{C}_{\text{new}} \neq \mathcal{C}$ **then**
        Update the local tree, i.e., add $\mathcal{C}_{\text{new}}$ to $\mathcal{T}$
        Output (SUCCESS, sid, 1) to P.
      else
        Output (SUCCESS, sid, 0) to P.
      On response (CONTINUE, sid) send (MULTICAST, sid, $\mathcal{C}_{\text{new}}$) to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$. ▷ Broadcast current chain
- Upon receiving (FETCH-NEW, sid) if do the following:
  Send (FETCH, sid) to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ and denote the response by (FETCH, sid, b).
  Extract all valid chains $\mathcal{C}_1, \ldots, \mathcal{C}_k$ from b and add them to $\mathcal{T}$.
  Extract states $\vec{\mathtt{st}}_1, \ldots, \vec{\mathtt{st}}_k$ from $\mathcal{C}_1, \ldots, \mathcal{C}_k$ and output them.

Functionality $\mathcal{F}_{\text{StX}}^{\Delta,p_H,p_A}$

The functionality is parametrized with a set of parties $\mathcal{P}$. Any newly registered (resp. deregistered) party is added to (resp. deleted from) $\mathcal{P}$. For each party $P \in \mathcal{P}$ the functionality manages a tree $\mathcal{T}_P$ where each rooted path corresponds to a valid state the party has received. Initially each tree contains the genesis state $\texttt{gen}$.
Finally, it manages a buffer $\vec{M}$ which contains successfully submitted states which have not yet been delivered to (some) parties in $\mathcal{P}$.

Submit/receive new states:

- Upon receiving $(\text{SUBMIT-NEW}, \text{sid}, \vec{\texttt{st}}, \texttt{st})$ from some participant $P_s \in \mathcal{P}$, if $\mathsf{isvalidstate}_{\mathbb{P}}(\vec{\texttt{st}}||\texttt{st}) = 1$ and $\vec{\texttt{st}} \in \mathcal{T}_P$ do the following:

  1. Sample B according to a Bernoulli-Distribution with parameter p_H (or p_A if P_s is dishonest).
  2. If $B = 1$, set $\vec{\texttt{st}}_{new} \leftarrow \vec{\texttt{st}}||\texttt{st}$ and add $\vec{\texttt{st}}_{new}$ to $\mathcal{T}_{P_s}$.
     Else set $\vec{\texttt{st}}_{new} \leftarrow \vec{\texttt{st}}$.
  3. Output $(\text{SUCCESS}, \text{sid}, B)$ to P_s.
  4. On response $(\text{CONTINUE}, \text{sid})$ where $\mathcal{P} = \{P_1, \ldots, P_n\}$ choose n new unique message-IDs $\text{mid}_1, \ldots, \text{mid}_n$, initialize n new variables $D_{\text{mid}_1} := D_{\text{mid}_1}^{MAX} := \ldots := D_{\text{mid}_n} := D_{\text{mid}_n}^{MAX} := 1$ set $\vec{M} := \vec{M}||(\vec{\texttt{st}}_{new}, \text{mid}_1, D_{\text{mid}_1}, P_1)|| \ldots ||(\vec{\texttt{st}}_{new}, \text{mid}_n, D_{\text{mid}_n}, P_n)$, and send $(\text{SUBMIT-NEW}, \text{sid}, \vec{\texttt{st}}_{new}, P_s, (P_1, \text{mid}_1), \ldots, (P_n, \text{mid}_n))$ to the adversary.

- Upon receiving $(\text{FETCH-NEW}, \text{sid})$ from a party $P \in \mathcal{P}$ or $\mathcal{A}$ (on behalf of P), do the following:

  1. For all tuples $(\vec{\texttt{st}}, \text{mid}, D_{\text{mid}}, P) \in \vec{M}$ set $D_{\text{mid}} := D_{\text{mid}} - 1$.
  2. Let $\vec{M}_0^P$ denote the subvector of $\vec{M}$ including all tuples of the form $(\vec{\texttt{st}}, \text{mid}, D_{\text{mid}}, P)$ where $D_{\text{mid}} = 0$ (in the same order as they appear in $\vec{M}$). For each tuple $(\vec{\texttt{st}}, \text{mid}, D_{\text{mid}}, P) \in \vec{M}_0^P$ add $\vec{\texttt{st}}$ to $\mathcal{T}_P$. Delete all entries in $\vec{M}_0^P$ from $\vec{M}$ and send $\vec{M}_0^P$ to P.

- Upon receiving $(\text{SEND}, \text{sid}, \vec{\texttt{st}}, P')$ from $\mathcal{A}$ on behalf some *corrupted* $P \in \mathcal{P}$, if $P' \in \mathcal{P}$ and $\vec{\texttt{st}} \in \mathcal{T}_P$, choose a new unique message-ID mid, initialize $D := 1$, add $(\vec{\texttt{st}}, \text{mid}, D_{\text{mid}}, P')$ to $\vec{M}$, and return $(\text{SEND}, \text{sid}, \vec{\texttt{st}}, P', \text{mid})$ to $\mathcal{A}$.

Further adversarial influence on the network:

- Upon receiving $(\text{SWAP}, \text{sid}, \text{mid}, \text{mid}')$ from $\mathcal{A}$, if mid and mid$'$ are message-IDs registered in the current $\vec{M}$, swap the corresponding tuples in $\vec{M}$. Return $(\text{SWAP}, \text{sid})$ to $\mathcal{A}$.

- Upon receiving $(\text{DELAY}, \text{sid}, T, \text{mid})$ from $\mathcal{A}$, if T is a valid delay, mid is a message-ID for a tuple $(\vec{\texttt{st}}, \text{mid}, D_{\text{mid}}, P)$ in the current $\vec{M}$ and $D_{\text{mid}}^{MAX} + T \leq \Delta$, set $D_{\text{mid}} := D_{\text{mid}} + T$ and set $D_{\text{mid}}^{MAX} := D_{\text{mid}}^{MAX} + T$.

Figure 8.9: The state exchange functionality. Parameters are the delay Δ and the success probabilities p_H and p_A for honest and adversarial submissions.

Lemma 8.6.1. *Let $p := \frac{p}{2^\kappa}$. The protocol StateExchange-Protocol$_{q,p}$ UC-realizes functionality $\mathcal{F}_{\text{StX}}^{\Delta,p_H,p_A}$ in the $(\mathcal{F}_{\text{RO}}, \mathcal{F}_{\text{N-MC}}^{\Delta})$-hybrid model where $p_A := p$ and $p_H := 1-(1-p)^q$.*

Proof. We consider the following simulator:

Simulator $\mathcal{S}_{\text{stx}}$

Initialization:

Set up a tree of valid chains $\mathcal{T} \leftarrow \{(\mathbf{G})\}$ and an empty network buffer $\vec{M}$.
Set up an empty random oracle table H and set $H[\mathbf{G}]$ to a uniform random value in $\{0,1\}^\kappa$. If the simulator ever tries to add a colliding entry to H, abort with COLLISION-ERROR.
The simulator manages a set $\mathcal{P}_{RO}$ of parties registered to the random oracle and a set of parties $\mathcal{P}_{net}$ registered to the network.

Simulating the Random Oracle:

- Upon receiving $(\text{EVAL}, \text{sid}, v)$ for $\mathcal{F}_{\text{RO}}$ from $\mathcal{A}$ on behalf of corrupted $P \in \mathcal{P}_{RO}$ do the following.
  1. If $H[v]$ is already defined, output $(\text{EVAL}, \text{sid}, v, H[v])$.
  2. If v is of the form $(\mathtt{s}, \mathtt{st}, \mathtt{n})$ and there exists[a] a chain $\mathcal{C} = \mathbf{B}_1, \ldots, \mathbf{B}_n$ such that $H[\mathbf{B}_n] = \mathtt{s}$ proceed as follows. If $\mathcal{C} \notin \mathcal{T}$ abort with TREE-ERROR. Otherwise continue. Extract the state $\vec{\mathtt{st}}$ from $\mathcal{C}$ and extract the state block $\mathtt{st}$ from v. Send $(\text{SUBMIT-NEW}, \text{sid}, \vec{\mathtt{st}}, \mathtt{st})$ to $\mathcal{F}_{\text{StX}}$ and denote by $(\text{SUCCESS}, B)$ the output of $\mathcal{F}_{\text{StX}}$. If $B = 1$ set $H[v]$ to a uniform random value in $\{0,1\}^\kappa$ strictly smaller[b] than $\mathtt{D}$. Add $\mathcal{C}||v$ to $\mathcal{T}$. Otherwise set $H[v]$ to a uniform random value in $\{0,1\}^\kappa$ larger than $\mathtt{D}$. Output $(\text{EVAL}, \text{sid}, v, H[v])$.
  3. Otherwise set v to a uniform random value in $\{0,1\}^\kappa$ and output $(\text{EVAL}, \text{sid}, v, H[v])$.

Simulating the Network:

- Upon receiving $(\text{MULTICAST}, \text{sid}, (m_{i_1}, P_{i_1}), \ldots, (m_{i_\ell}, P_{i_\ell}))$ for $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ from $\mathcal{A}$ on behalf of corrupted $P \in \mathcal{P}_{net}$ with $\{P_{i_1}, \ldots, P_{i_\ell}\} \subseteq \mathcal{P}_{net}$ proceed as follows.
  1. Choose ℓ new unique message-IDs $\text{mid}_{i_1}, \ldots, \text{mid}_{i_\ell}$, initialize ℓ new variables $D_{\text{mid}_{i_1}} := \ldots := D_{\text{mid}_{i_\ell}} := 1$, set $\vec{M} := \vec{M}||(m_{i_1}, \text{mid}_{i_1}, D_{\text{mid}_{i_1}}, P_{i_1})|| \ldots ||(m_{i_\ell}, \text{mid}_{i_\ell}, D_{\text{mid}_{i_\ell}}, P_\ell)$.
  2. For each (m_{i_j}, P_{i_j}) where m_{i_j} is a chain in $\mathcal{T}$ extract the state $\vec{\mathtt{st}}_{i_j}$ from m_{i_j}, and send $(\text{SEND}, \text{sid}, \vec{\mathtt{st}}, P_{i_j})$ to $\mathcal{F}_{\text{StX}}$. Store the message-ID $\widehat{\text{mid}_{i_j}}$ returned by $\mathcal{F}_{\text{StX}}$ with mid_{i_j}. Note that if P has not yet received that state, it is first fetched by $\mathcal{A}$ on behalf of P and if an unknown state is encoded, a random oracle query is simulated for the input to simulate the chain's validity and its possible inclusion into $\mathcal{T}$.
  3. Output $(\text{MULTICAST}, \text{sid}, (m_{i_1}, P_{i_1}, \text{mid}_{i_1}), \ldots, (m_{i_\ell}, P_{i_\ell}, \text{mid}_{i_\ell}))$ to $\mathcal{A}$.

- Upon receiving $(\text{FETCH}, \text{sid})$ for $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ from $\mathcal{A}$ on behalf of corrupted $P \in \mathcal{P}_{net}$ proceed as follows.

  1. For all tuples $(m, \text{mid}, D_{\text{mid}}, P) \in \vec{M}$, set $D_{\text{mid}} := D_{\text{mid}} - 1$.
  2. Let $\vec{M}_0^P$ denote the subvector $\vec{M}$ including all tuples of the form $(m, \text{mid}, D_{\text{mid}}, P)$ with $D_{\text{mid}} = 0$ (in the same order as they appear in $\vec{M}$). Delete all entries in $\vec{M}_0^P$ from $\vec{M}$, and send $\vec{M}_0^P$ to $\mathcal{A}$.

- Upon receiving a message $(\text{DELAYS}, \text{sid}, (T_{\text{mid}_{i_1}}, \text{mid}_{i_1}), \ldots, (T_{\text{mid}_{i_\ell}}, \text{mid}_{i_\ell}))$ do the following for each pair $(T_{\text{mid}}, \text{mid})$ in this message:

  1. If T_{mid} is a valid delay (i.e., it encodes an integer in unary notation) and mid is a message-ID registered in the current $\vec{M}$, set $D_{\text{mid}} := \max\{1, D_{\text{mid}} + T_{\text{mid}}\}$; otherwise, ignore this tuple.
  2. If the simulator knows a corresponding $\mathcal{F}_{\text{STX}}$-message-ID $\widehat{\text{mid}}$ for mid send $(\text{DELAY}, \text{sid}, T_{\text{mid}}, \widehat{\text{mid}})$ to $\mathcal{F}_{\text{STX}}$.

- Upon receiving a message $(\text{SWAP}, \text{sid}, \text{mid}_1, \text{mid}_2)$ from the adversary do the following:

  1. If mid_1 and mid_2 are message-IDs registered in the current $\vec{M}$, then swap the corresponding tuples in $\vec{M}$.
  2. If the simulator knows for both mid_1 and mid_2 $\mathcal{F}_{\text{STX}}$-message-IDs $\widehat{\text{mid}_1}$ and $\widehat{\text{mid}_2}$ send $(\text{SWAP}, \text{sid}, \widehat{\text{mid}_1}, \widehat{\text{mid}_2})$ to $\mathcal{F}_{\text{STX}}$.
  3. Output $(\text{SWAP}, \text{sid})$ to $\mathcal{A}$.

Interaction with the State Exchange Functionality :

- Upon receiving $(\text{SUBMIT-NEW}, \text{sid}, \vec{\mathtt{st}}, P_s, (P_1, \widehat{\text{mid}_1}), \ldots, (P_n, \widehat{\text{mid}_n}))$ from $\mathcal{F}_{\text{STX}}$ where $\vec{\mathtt{st}} = \mathtt{st}_1, \ldots, \mathtt{st}_k$ and $\{P_1, \ldots, P_n\} := \mathcal{P}_{net}$ proceed as follows

  1. If there exist a chain $\mathcal{C} \in \mathcal{T}$ with state $\vec{\mathtt{st}}$ generate new unique message-IDs $\text{mid}_1, \ldots, \text{mid}_n$, initialize $D_1 := \cdots := D_n = 1$, set $\vec{M}||(\mathcal{C}, \text{mid}_{i1}, D_{\text{mid}_1}, P_1)|| \ldots ||(\mathcal{C}, \text{mid}_n, D_{\text{mid}_n}, P_n)$, and store the message-IDs $\widehat{\text{mid}_i}$ along the message-IDs mid_i.
     Output $(\text{MULTICAST}, \text{sid}, \mathcal{C}, P_s, (P_1, \text{mid}_1), \ldots, (P_n, \text{mid}_n))$ to the adversary.
  2. Otherwise find a chain $\mathcal{C}'$ in $\mathcal{T}$ with state $\mathtt{st}_1, \ldots, \mathtt{st}_{k-1}$ [c]. Choose a random nonce $\mathtt{n}$ and set $\mathbf{B}_k = (H[\mathbf{B}_{k-1}], \mathtt{st}_k, \mathtt{n})$ and set $H[\mathbf{B}_k]$ to a uniform random value in $\{0,1\}^\kappa$ strictly smaller than $\mathtt{D}$. Add the chain $\mathcal{C} = \mathcal{C}'||\mathbf{B}_k$ to $\mathcal{T}$.
     Generate new unique message-IDs $\text{mid}_1, \ldots, \text{mid}_n$, initialize $D_1 := \cdots := D_n = 1$, set $\vec{M}||(\mathcal{C}, \text{mid}_{i1}, D_{\text{mid}_1}, P_1)|| \ldots ||(\mathcal{C}, \text{mid}_n, D_{\text{mid}_n}, P_n)$, and store the message-IDs $\widehat{\text{mid}_i}$ along the message-IDs mid_i. Output $(\text{MULTICAST}, \text{sid}, \mathcal{C}, P_s, (P_1, \text{mid}_1), \ldots, (P_n, \text{mid}_n))$ to the adversary.

[a] This can be checked efficiently using H under the assumption that there are no collisions.

[b] Can be done efficiently using rejection sampling.

[c] Such a chain must exist as $\mathtt{st}_1, \ldots, \mathtt{st}_{k-1}$ is a successfully submitted state in $\mathcal{F}_{\text{StTX}}$ in which case the simulator knows a corresponding chain.

The proof works similar as the one for Lemma 5.1 in [PSS17]. Recall the notation from Section 2.4 and introduce the shorthand notation $T_{\text{real}} := T_{\text{EXEC}_{\textsf{StateExchange-Protocol},\mathcal{A},\mathcal{Z}}(\kappa,z)}$ which is the (distribution of the) joint view of all parties in the execution of StateExchange-Protocol for adversary $\mathcal{A}$ and environment $\mathcal{Z}$ (upon some input z). Denote by $T_{\text{ideal}} := T_{\text{EXEC}_{\mathcal{F}_{\text{StTX}},\mathcal{S}_{\text{stx}},\mathcal{Z}}}(\kappa, z)$ the joint view of all parties for $\mathcal{F}_{\text{StTX}}$ with simulator $\mathcal{S}_{\text{stx}}$. In the following, we treat the arguments κ and z as implicit.

Define a new hybrid world, via the following random experiment: the experiment is defined as the real-world execution except that the random oracle aborts on collisions with COLLISION-ERROR and that adversarial oracle queries are emulated as in $\mathcal{S}_{\text{stx}}$. We use the shorthand $\text{HYB}_{\mathcal{A},\mathcal{Z}}$ to refer to this hybrid world (defined analogously to $\text{EXEC}_{\cdot,\cdot,\mathcal{Z}}$). The only difference is thus that in the hybrid world we may abort with COLLISION-ERROR or TREE-ERROR as in the ideal execution. Let T_{hyb} be the associated distribution of the joint view.

Let event1 be the event that some parties query two different values v, v' such that $H[v] = H[v']$, i.e. the event that a hash-collision occurs (this event is a condition on the realized transcript tr in the support of T_{real} or T_{hyb}, respectively). For any two queries the probability that they return the same hash value is $2^{-\kappa}$. By a union bound over all queries we have that event1 happens with probability at most $\mathsf{poly}(\kappa) \cdot 2^{-\kappa}$ in both worlds. Note that if event1 does not happen the hybrid random experiment does not abort with COLLISION-ERROR.

Let event2 be the event that some party makes a query $H[(\mathtt{s}, \cdot, \cdot)]$ where no v exists such that $H[v] = \mathtt{s}$, but later some party makes a query v' such that $H[v'] = \mathtt{s}$. The probability that any query $H[(\mathtt{s}, \cdot, \cdot)]$ a later query returns $\mathtt{s}$ is $2^{-\kappa}$ in both worlds By a union bound over all queries we have that event2 happens with probability at most $\mathsf{poly}(\kappa) \cdot 2^{-\kappa}$ in both worlds.

Next, we show that the TREE-ERROR abort does not occur in the

hybrid world execution conditioned under event1 and event2 not happening. Assume for contradiction that $\text{HYB}_{\mathcal{A},\mathcal{Z}}$ aborts with TREE-ERROR with event1 and event2 not happening. Let $\mathcal{C} = \mathbf{B}_1, \ldots, \mathbf{B}_n$ be the shortest valid chain created in the experiment $\text{HYB}_{\mathcal{A},\mathcal{Z}}$ such that $\mathbf{B}_1, \ldots, \mathbf{B}_{n-1} \in \mathcal{T}$ but $\mathbf{B}_1, \ldots, \mathbf{B}_n \notin \mathcal{T}$. Let $\mathbf{B}_i = (\mathtt{s}_i, \mathtt{st}_i, \mathtt{n}_i)$. Since $\mathcal{C}$ is a valid chain we have $H[(\mathtt{s}_n, \mathtt{st}_n\mathtt{n}_n)] < D$. But at the time $\mathbf{B}_n$ was added to H no valid chain existed where the last block has hash value $\mathtt{s}_n$ (otherwise $\mathcal{C}$ would be in $\mathcal{T}$). This implies that no earlier query to H could have returned $\mathtt{s}_n$, since if the query was $\mathbf{B}_{n-1}$ $\mathcal{C}$ would not be the shortest chain with the above property and if the query was not $\mathbf{B}_{n-1}$ the event event1 must have happened. This implies that event2 must have happened, which is a contradiction.

This implies that conditioned under event1 and event2 not happening, the hybrid-world execution proceeds the same as the real-world execution and hence the two worlds are statistically close with respect to efficient environments $\mathcal{Z}$, i.e., $\text{EXEC}_{\mathsf{StateExchange\text{-}Protocol},\mathcal{A},\mathcal{Z}} \approx \text{HYB}_{\mathcal{A},\mathcal{Z}}$.

Now we compare $\text{HYB}_{\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\mathcal{F}_{\text{STX}},\mathcal{S}_{\text{stx}},\mathcal{Z}}$. Consider the event where a honest miner queries a block $(\mathtt{s}, \mathtt{st}, \mathtt{n})$ and fails, i.e. where $H[(\mathtt{s}, \mathtt{st}, \mathtt{n})] > \mathtt{D}$. In the hybrid execution, this query is stored in the random oracle table while the simulator in the ideal world does not store the query in the random oracle table. Under the condition that such failed queries are not repeated, the hybrid-world execution and the ideal-world execution proceed in identical ways (note that the network simulation in $\mathcal{S}_{\text{stx}}$ perfectly mimics the real and the hybrid worlds).

Note that the nonce $\mathtt{n}$ in a 'failed' query $(\mathtt{s}, \mathtt{st}, \mathtt{n})$ is choosen uniform at random from $\{0,1\}^\kappa$ by honest parties. This implies that with probability $\mathsf{poly}(\kappa) \cdot 2^{-\kappa}$ it was never queried before. As honest miner discard 'failed' queries (and failed queries do not leave the ITI and hence are hidden from the adversary) it also follows that except with probability $\mathsf{poly}(\kappa) \cdot 2^{-\kappa}$ the query will not be queried again (by any honest or corrupted party) unless the nonce of that failed query would be successfully guessed. By a union bound over all failed queries we have that failed queries are never queried twice except with probability $\mathsf{poly}(\kappa) \cdot 2^{-\kappa}$. Thus, $\text{EXEC}_{\mathcal{F}_{\text{STX}},\mathcal{S}_{\text{stx}},\mathcal{Z}} \approx \text{HYB}_{\mathcal{A},\mathcal{Z}}$. This concludes the proof. □

Proving the Modularization of the Ledger-Protocol

We present the modularized UC Bitcoin protocol in Appendix B.3. We have the following lemma:

Lemma 8.6.2. *The UC Bitcoin protocol* $\mathsf{Ledger\text{-}Protocol}_{q,\mathcal{D},T}$ *UC emulates* $\mathsf{Modular\text{-}Ledger\text{-}Protocol}_T$ *that runs in a hybrid world with access to the functionality* $\mathcal{F}_{\text{StX}}^{\Delta,p_H,p_A}$ *with* $p_A := \frac{\mathcal{D}}{2^\kappa}$ *and* $p_H = 1-(1-p_A)^q$*, and where* Δ *denotes the upper bound on the network delay.*

Proof. The proof involves a sequence of modifications, morphing from the original protocol to the modularized protocol in a "game-hopping" style and finally identifying the sub-process $\mathsf{StateExchange\text{-}Protocol}$ that can be replaced by calls to $\mathcal{F}_{\text{StX}}$ and concluding the statement by invoking Lemma 8.6.1. The detailed proof is given in Appendix B.3.2. □

8.6.3 Second Proof Step

We now proof that if honest parties have some advantage over the dishonest parties in winning the lottery, then the UC Bitcoin protocol $\mathsf{Modular\text{-}Ledger\text{-}Protocol}_T$ realizes the ledger functionality. By the composition theorem, we can directly conclude that $\mathsf{Ledger\text{-}Protocol}_{q,\mathtt{D},T}$ realizes the Bitcoin ledger functionality.

Relevant Quantities of the Analysis

The main theorem will require a condition on the power of the adversary and it is useful to describe here the random variables induced by a pair $(\mathcal{Z},\mathcal{A})$. Recall from Sections 8.4.2 and 8.4.3 that a party is honest-and-synchronized if it either joined at the onset of the execution or it joined a sufficient number of rounds ago (depending on the delay). Furthermore, recall that a logical round consists of two clock-ticks. In the following, we denote the round number by r (which consists of two mini-rounds).

Definition 8.6.3 (Query Power)**.** We define for the real-world execution of $\mathsf{Modular\text{-}Ledger\text{-}Protocol}_T$ with respect to the pair $(\mathcal{Z},\mathcal{A})$ the sequence of random variables $Q_H^{(r)}$ to measure the number of distinct honest-and-synchronized parties that are activated in the working mini-round of round r to submit a query to $\mathcal{F}_{\text{StX}}^{\Delta,p_H,p_A}$. Analogously, denote by $Q_A^{(r)}$ the

number of submit-queries to $\mathcal{F}_{\text{StX}}^{\Delta,p_H,p_A}$ from corrupted parties in round r, and by $Q_{H,DS}^{(r)}$ the number submit-queries by honest-but-desynchronized parties in the working mini-round of round r.

Definition 8.6.4 (Mining Power.)**.** We define mining power as simple functions of the query-power. Note that in our analysis, p_A and p_H are constants. We have:

- The total mining power $\mathtt{T}_{mp}^{(r)} := Q_A^{(r)} \cdot p_A + (Q_H^{(r)} + Q_{H,DS}^{(r)}) \cdot p_H$.
- The adversarial mining power $\beta^{(r)} := p_A \cdot Q_A^{(r)} + p_H \cdot Q_{H,DS}^{(r)}$.
- The honest mining power $\alpha^{(r)} := 1 - (1 - p_H)^{Q_H^{(r)}}$.

It might be useful to recall that from Bernoulli's inequality we have $\alpha^{(r)} \leq p_H \cdot Q_H^{(r)}$. For small values of p_H (as usual in Bitcoin) this upper bound is a good approximation of $\alpha^{(r)}$.

Note that $\alpha^{(r)}$, $\beta^{(r)}$, and $\mathtt{T}_{mp}^{(r)}$ are random variables (on integer domains). For example. $\alpha^{(r)}$ maps the number of honest-and-synchronized submit-queries to the probability that at least one is a successful query. More formally, conditioned on $Q_H^{(r)} = q$, the random variable $\alpha^{(r)}$ is the probability of at least one success among q queries and the expected value of $\alpha^{(r)}$ corresponds to the probability of at least one successful state-extension in round r of the execution. The reason is that $\mathcal{F}_{\text{StX}}^{\Delta,p_H,p_A}$ treats each submit-query independently at random. This is the main motivation to introduce this intermediate step.

The analysis

In the analysis of Bitcoin, conditions are needed that allow to reasonably lower and upper bound expected values of the above random variables (and their variances). As we will quickly recap below, it is shown in [PSS17] that if the involved query power exceeds any limits in the constant-difficulty case, then no security guarantees can be obtained. We start with the following definition.

Definition 8.6.5 (Query and Mining Pattern)**.** We say that the pair $(\mathcal{Z}, \mathcal{A})$, running for R rounds (referred to by numbers $0, \ldots, R-1$) obeys

the query pattern $(\vec{h}, \vec{a}, \vec{d})$ if, for any round r, we have

$$Q_H^{(r)} \geq h_r, \quad Q_A^{(r)} \leq a_r, \quad Q_{H,DS}^{(r)} \leq d_r$$

where $\vec{h} = (h_0, \ldots, h_{R-1}), \vec{a} = (a_0, \ldots, a_{R-1}), \vec{d} = (d_0, \ldots, d_{R-1})$ are vectors consisting of positive integers. Consequently, the pair $(\mathcal{Z}, \mathcal{A})$ obeys the associated mining pattern denoted by $(\vec{\alpha}, \vec{\beta})$, where vectors $\vec{\alpha} = (\alpha_0, \ldots, \alpha_{R-1})$ and $\vec{\beta} = (\beta_0, \ldots, \beta_{R-1})$ are defined by the mapping

$$\alpha^{(r)} \geq 1 - (1 - p_H)^{h_r} =: \alpha_r$$
$$\beta^{(r)} \leq p_A \cdot a_r + p_H \cdot d_r =: \beta_r.$$

Technically, these definitions imply lower and upper bounds on the expectations of the random variables $\alpha^{(r)}$ and $\beta^{(r)}$ respectively, which is what will be eventually needed.

Definition 8.6.6 (Power Limits)**.** The pair $(\mathcal{Z}, \mathcal{A})$ is said to be q_{tot}-query-limited if $Q_H^{(r)} + Q_A^{(r)} + Q_{H,DS}^{(r)} \leq q_{tot}$. The pair $(\mathcal{Z}, \mathcal{A})$ is said to be $\mathtt{T}_{mp}$-mining limited if for all r,

$$\mathtt{T}_{mp}^{(r)} \leq \mathtt{T}_{mp}.$$

The bounds in the theorem will depend on several worst-case quantities that we introduce below.

Definition 8.6.7. For mining patterns $(\vec{\alpha}, \vec{\beta})$, we use the shorthand notation

$$\alpha_{min} := \min\{\alpha_r\}_{r\in[0,R-1]} \quad \text{and} \quad \alpha_{max} := \max\{\alpha_r\}_{r\in[0,R-1]};$$
$$\beta_{min} := \min\{\beta_r\}_{r\in[0,R-1]} \quad \text{and} \quad \beta_{max} := \max\{\beta_r\}_{r\in[0,R-1]}.$$

For a (non-empty) subset $S \subseteq \{0, \ldots, R-1\}$ of rounds we define the corresponding averages by

$$\overline{\alpha}_S := \frac{1}{|S|} \cdot \sum_{r\in S} \alpha_r \quad \text{and} \quad \overline{\beta}_S := \frac{1}{|S|} \cdot \sum_{r\in S} \beta_r.$$

For $\mathtt{T}_{mp}$-mining limited pairs $(\mathcal{Z}, \mathcal{A})$, we define the relative-power fractions

$$\rho_h := \frac{\alpha_{min}}{\mathtt{T}_{mp}} \quad \text{and} \quad \rho_a := \frac{\beta_{min}}{\mathtt{T}_{mp}}.$$

We call a subset S of rounds an interval if it consists of consecutive round numbers $r, \ldots, r+t$ for some integers $r, t \geq 0$.

Following [PSS17], the theorem will take into account that the network delay Δ decreases the effectiveness of the actual honest mining power:

Definition 8.6.8 (Discount function.)**.** We define the function $\gamma(\alpha, \Delta) := \frac{\alpha}{1+\alpha\Delta}$ for $\alpha, \Delta > 0$.

We are now ready to state and prove the main theorem which assures that we can realize the ledger for a given range of parameters.

Theorem 8.6.9. *Let $p \in (0,1)$, integer $q \geq 1$, $p_H = 1-(1-p)^q$, and $p_A = p$. Let $\Delta \geq 1$ be the upper bound on the network delay. For all pairs $(\mathcal{Z}, \mathcal{A})$ running for R rounds which obey the $(\vec{\alpha}, \vec{\beta})$ mining pattern as of Definition 8.6.5 and which are τ_{mp}-limited as of Definition 8.6.6, the real-world execution of protocol $\mathsf{Modular\text{-}Ledger\text{-}Protocol}_T$ (in the $(\mathcal{G}_{\text{CLOCK}}, \mathcal{F}_{\text{StX}}^{\Delta,p_H,p_A}, \mathcal{F}_{\text{N-MC}}^{\Delta})$-hybrid world) is indistinguishable from the ideal-world execution with ledger functionality $\mathcal{G}_{\text{LEDGER}}^{\mathbb{P}}$ (and the simulator defined in the proof), if for some $\lambda > 1$, it holds that for any interval S of rounds of size $t \geq 1$ and any $S' \subseteq S$ of size $t' \in [\max\{1, t \cdot (1 - \Delta\alpha_{max})\}, \ldots, t]$ the relation*

$$\overline{\alpha}_{S'} \cdot (1 - 2 \cdot (\Delta + 1) \cdot \tau_{mp}) \geq \lambda \cdot \overline{\beta}_S \tag{8.1}$$

holds, and if the ledger parameters (which are positive and integer-valued) satisfy the conditions

$$\mathtt{windowSize} = T \quad \text{and} \quad \mathtt{Delay} = 4\Delta,$$

$$\mathtt{maxTime}_{\mathtt{window}} \geq \frac{2 \cdot \mathtt{windowSize}}{(1-\delta) \cdot \gamma_{min}} \quad \text{and} \quad \mathtt{minTime}_{\mathtt{window}} \leq \frac{2 \cdot \mathtt{windowSize}}{(1+\delta) \cdot \mathtt{T}_{mp}},$$

$$\eta \geq \min\{(1+\delta) \cdot \frac{\beta_{max}}{\gamma_{min}} \cdot \mathtt{windowSize}, \mathtt{windowSize}\},$$

where the quantities are defined as in Definition 8.6.7 and where $\gamma_{min} := \gamma(\alpha_{min}, \Delta)$ and $\delta > 0$ is an arbitrary constant. In particular, the realization is perfect except with probability $R \cdot negl(T)$, where $negl(T)$ denotes a negligible function in T.

Remarks. Before proving the theorem, it is instructive to recall the flat model of Bitcoin and to see how the above quantities appear there. By the above definitions and theorem statement, we see that we only make statements if the honest mining power is not too small, the dishonest mining power is not too large (and stands in a certain relation to the honest mining power) and if the respective mining power values are in a reasonable range to the overall mining power. In particular, the theorem expresses a condition that the average honest mining power dominates the average mining power of the adversary, even if the honest average is taken over slightly smaller intervals (note that in particular, for each singleton set S, we obtain that the familiar condition that α_r should dominate β_r). Note that β_{min} is the most restrictive restriction (but not a lower-bound) on the adversary (similarly, α_{max} is the best guaranteed lower-bound for honest-and-synchronous mining power). In general, the adversary (and hence the environment) is free to activate as many ITI's unless it would exceed $\mathtt{T}_{mp}$ if the environment is $\mathtt{T}_{mp}$-bounded, and no more than what is allowed by $\vec{\beta}$. This is a more general setting in the fixed-difficulty setting compared to previous works. We show how to cast previous analyses (with respect to fixed difficulty) in our setting in Section 8.7.1. Furthermore, we show in the next subsection how to get a better bound for chain-quality.

Looking ahead, for example in [PSS17], the overall number of parties is fixed to be some number n and there is an upper bound on the number of dishonest parties ρn (and de-synchronized parties are not allowed by definition). Assume for simplicity that $p_H = p_A = p$ for a very small value $p > 0$. We then obtain $\alpha_{min} \approx (1 - \rho) \cdot n \cdot p$ and $\beta_{max} \approx \rho_H \cdot n \cdot p$. By $\mathtt{T}_{mp} = n \cdot p$ and since the mining pattern as defined above is flat in flat models (cf. Section 8.7.1), the correspondence $\rho_a = \rho$ and $\rho_h = (1 - \rho)$ follows.

Also, as pointed out by [PSS17], for too large values of p in a range that would yield $\mathtt{T}_{mp} = n \cdot p > \frac{1}{\Delta}$ (where Δ is the network delay), there is an attack against the protocol, even if one assumes an honest majority. This indicates that the main condition of the theorem in equation (8.1) is also necessary up to a constant factor.

We now prove our main theorem.

Proof of Theorem 8.6.9. We start with an overview followed by a sequence of claims.

Overview. We prove the theorem by proving the security with respect to the so-called EUC notion (externalized UC) with the global clock as the only shared functionality. This then not only implies standard UC realization (with respect to a local clock), but also implies the full GUC statement by the equivalence shown in [CDPW07b]. In order to show the theorem we specify the simulator for the ideal world $\mathcal{S}_{\text{ledg}}$. $\mathcal{S}_{\text{ledg}}$ is specified as pseudo-code in Appendix B.4. Let us explain the general structure: the simulator internally runs the round-based mining procedure of every honest party. Whenever a working mini-round is over, i.e., whenever the real world parties have issued their queries to $\mathcal{F}_{\text{StX}}$, then the simulator will assemble the views of its simulated honest-and-synchronized miners and determine their common prefix of states, which is the longest state stored or received by each simulated party when chopping off T blocks. The adversary will then propose a new block candidate, i.e., a list of transactions, to the ledger to announce that the common prefix has increased (procedure ExtendLedgerState). The ledger will apply the Blockify on this list of transactions and add it to the state. Note that since Blockify does not depend on time, the current time of the ledger has no influence on this output. To reflect that not all parties have the same view on this common prefix, the simulator can adjust the state pointers accordingly (procedure AdjustView). The simulation inside the simulator is perfect and is simply the emulation of real-world processes. What restricts a perfect simulation is the requirement of a consistent prefix and the restrictions imposed by ExtendPolicy. In order to show that these restrictions are not forbidding a proper simulation, we have to justify, why the choice of the parameters in the theorem are sufficient to guarantee that (except with negligible probability). To this end, we analyze the real-world execution to bound the corresponding bad events that prevent a perfect simulation.

We basically follow the proof ideas of Pass, Seeman, and shelat [PSS17] to bound the bad events and adapt their observations to our setting. The analysis is divided into several different claims about the real-world execution. They include properties such as a lower-bound on the chain growth, the chain quality, or an upper-bound on the chain growth. These claims show that our simulator can simulate the real-world view perfectly, since the restrictions imposed by the ledger prohibit that only with negligible probability, where the distinguishing advantage is upper bounded by $R \cdot \text{negl}(T)$, where R denotes the number of rounds the protocol is running

and negl(·) denotes a negligible function in the parameter T.

Recall that each round consists of two time-ticks. Hence, if a statement is expressed with respect to a certain number t of rounds, it can equivalently be expressed with respect to $2t$ clock-ticks. Recall that the ledger parameters have to be given with respect to the clock, since the clock is the formal reference point of time. However, for the analysis, it is easier to think in rounds. In the following sections, if we refer to an interval $r, \ldots, r+t$, this refers to t *full* rounds, i.e., the time window when the clock first switched to the value $\tau = 2r$ up to the point when the clock value is $\tau \in \{2(r+t), 2(r+t)+1\}$.

Chain dissemination. We first state an obvious useful fact about the protocol's operation.

Lemma 8.6.10 (State dissemination)**.** *Let P_i and P_j be miners, and let $r \geq 0$. Assume P_i is honest in round r, and its adopted state has length ℓ. For any honest miner P_j in round $r + \Delta$ who registered to the network before round r, it holds that its adopted state must have at least length ℓ.*

Proof. By assumption, all messages, and in particular transmitted states of honest miners, are delayed maximally by Δ rounds. Thus, if such a miner receives a state of length ℓ, then any other honest miner will receive this state within the next Δ rounds since the protocol relays its adopted state. Additionally, if an honest miner successfully extends a ledger-state in round r, the new state is fetched by other honest miner at latest after Δ rounds if they were registered before round r. Hence by then, they will have adopted a chain of length at least ℓ. □

Probably the most useful corollary which is used in the sequel, is to apply the above lemma to the sub-class of honest-and-synchronized miners. Note that if P_j in the above lemma is honest-and-synchronized at round $r + \Delta$ it must have been registered to the network not later than at round $\max\{0, r - \Delta\}$ and hence the statement applies.

Analyzing chain growth. We now state the relation between time (measured in number of rounds) and guaranteed number of new state blocks.

Lemma 8.6.11 (Chain growth)**.** *Consider the real-world execution (under the conditions of the theorem). Let P_i be a miner, and let $r \geq 0$. Assume P_i is honest-and-synchronized in round r, and the (longest) state adopted by P_i in round r has length ℓ. Then, in round $r + t$, it holds that for any $\delta > 0$, except with probability $R \cdot negl(T)$, the length of the (longest) state adopted of any honest-and-synchronized miner P_j in that round has length at least $\ell + T$ if $t \geq \frac{T}{(1-\delta)\cdot\gamma_{min}}$.*

More generally, for an interval of rounds $r, \ldots, r+t$, we can guarantee a length increase of $\gamma \cdot t$ with $\gamma := \frac{\tau}{1+\tau\Delta}$ if for all possible subsets S of rounds of size $t' = t(1 - \gamma\Delta)$ of this interval we have $\overline{\alpha}_S \geq \tau$. The guarantee holds except with probability $\exp(-\Omega(t\gamma))$.

Proof. We first prove that for any real-world adversary $\mathcal{A}$, there is an adversary $\mathcal{A}'$ that, starting at the given round r, maximally delays messages and prove that in a real-world execution with $\mathcal{A}'$ the expected state length of an honest-and-synchronized miner in round $r+t$, where the expectation is taken over the randomness of the adversarial strategy, is no larger than with adversary $\mathcal{A}$ in round $r + t$. Given adversary $\mathcal{A}$, the adversary $\mathcal{A}'$ works as follows. It internally runs $\mathcal{A}$ until and including round r without any modifications. After round r, $\mathcal{A}'$ first delays all current messages in the network to the maximally possible delay. Also, after round r, whenever an honest-and-synchronized party sends a message containing a state, $\mathcal{A}'$ sets the maximal delay Δ for this message. Message delays defined by $\mathcal{A}$ for messages that contain valid states of honest parties are ignored. The adversary further ignores any message sent by $\mathcal{A}$ on behalf of corrupted parties after round r.

We define the following "hybrid world", which equals the real world execution, but with fixed randomness as follows: for random strings σ, σ', we define $\mathrm{HYB}_{\mathcal{F}_{\mathrm{StX}}(\sigma'),\mathcal{A}(\sigma),\mathcal{Z}}$ to be defined analogously to $\mathrm{EXEC}_{\cdot,\cdot,\mathcal{Z}}$ but where the internal coins of $\mathcal{A}$ and $\mathcal{F}_{\mathrm{StX}}$ are fixed to σ and σ' respectively (note that both are poly-bounded by the run-time restrictions of UC). Let $T^{\mathrm{hyb}}_{\mathcal{A}(\sigma),\mathcal{F}_{\mathrm{StX}}(\sigma'),\mathcal{Z}}$ be the associated distribution of the joint view (induced by the random coins of $\mathcal{Z}$). Let $\mathrm{Len}^r_i(T)$ be the function that maps a transcript T (of real-world and hybrid-world executions) to the length of the (longest) adopted chain by (honest-and-synchronized) miner i in round r.

We first give an inductive proof to show that for any $r > 0$, and all

strings σ, σ',

$$\Pr_{\sigma_Z \in_R \{0,1\}^{\mathsf{poly}(\kappa)}}[\mathrm{Len}_i^{r+t}(T^{\mathrm{hyb}}_{\mathcal{A}(\sigma),\mathcal{F}_{\mathrm{STX}}(\sigma'),\mathcal{Z}(\sigma_Z)}) \geq \mathrm{Len}_i^{r+t}(T^{\mathrm{hyb}}_{\mathcal{A}'(\mathcal{A}(\sigma)),\mathcal{F}_{\mathrm{STX}}(\sigma'),\mathcal{Z}(\sigma_Z)})] = 1.$$

Base Case(s): We give the base cases $t = 0$ and $= 1$ to already include the arguments for the general case below. We argue for any fixed σ_Z and show that the condition in the event cannot be violated. Since adversary $\mathcal{A}$ and $\mathcal{A}'$ behave identical up to and including round r, the length of the longest state known or received by any party is the same. The reason is that $\mathcal{A}'$ and $\mathcal{A}$ play exactly the same strategy when the randomness is fixed, since $\mathcal{A}'$ itself does not use additional random coins and thus case $t = 0$ follows. Furthermore, when the randomness σ' of $\mathcal{F}_{\mathrm{STX}}$ is fixed, a miner i in any round r' is successful, if and only if it is successful in round r' with adversary $\mathcal{A}'$. Thus, the condition for $t = 1$ would only be violated if player i receives a longer state in round $r + 1$. However, since $\mathcal{A}'$ maximally delays messages, if any state arrives in round $r + 1$ in the real execution with $\mathcal{A}'$, then it arrives no later than $r + 1$ in the real execution with $\mathcal{A}$. This concludes the base cases.

Induction Step: $t \to t + 1$*:* By the induction hypothesis, we have that the condition

$$\mathrm{Len}_i^{r+t}(T^{\mathrm{hyb}}_{\mathcal{A}(\sigma),\mathcal{F}_{\mathrm{STX}}(\sigma'),\mathcal{Z}(\sigma_Z)}) \geq \mathrm{Len}_i^{r+t}(T^{\mathrm{hyb}}_{\mathcal{A}'(\mathcal{A}(\sigma)),\mathcal{F}_{\mathrm{STX}}(\sigma'),\mathcal{Z}(\sigma_Z)})$$

holds with probability one. We argue that $\mathrm{Len}_i^{r+t+1}(\cdot) \geq \mathrm{Len}_i^{r+t+1}(\cdot)$ holds as well (on the above arguments) with probability one. Assume this was not the case, then by the above reasoning, it can only be due to miner i receiving a state in round $r + t + 1$ that would increase the value of $\mathrm{Len}_i^{r+t+1}(T^{\mathrm{hyb}}_{\mathcal{A}'(\mathcal{A}(\sigma)),\mathcal{F}_{\mathrm{STX}}(\sigma'),\mathcal{Z}(\sigma_Z)})$ but would not increase the value of the variable $\mathrm{Len}_i^{r+t+1}(T^{\mathrm{hyb}}_{\mathcal{A}(\sigma),\mathcal{F}_{\mathrm{STX}}(\sigma'),\mathcal{Z}(\sigma_Z)})$ (since the success of miner i in round $r + t + 1$ is fixed given σ'). By the same reasoning as above, since $\mathcal{A}'$ maximally delays delivery of new states, if any state arrives in round r in the real execution with $\mathcal{A}'$, then it arrives no later than r in the real execution with $\mathcal{A}$. This concludes the induction proof.

We note that the hybrid world, if we sample σ, σ' this yields the distribution $T_{\mathrm{EXEC}_{\pi,\mathcal{A}',\mathcal{Z}}}(\kappa, z)$ (for any fixed input z to the environment).

Let us abbreviate this by $T_{\text{real},\mathcal{A}'}$ to save on notation (and assuming the input z is hard-coded in the environment). Similarly, let us denote $T_{\text{real},\mathcal{A}}$ the distribution in an execution with $\mathcal{A}$.

By taking the expectation over σ, σ' (and by the law of total probability), we immediately get from the above arguments that for any positive integer c and any round r:

$$\begin{aligned}&\Pr[\text{Len}_i^{r+t}(T_{\text{real},\mathcal{A}}) \leq \text{Len}_i^r(T_{\text{real},\mathcal{A}}) + c] \\ &\leq \Pr[\text{Len}_i^{r+t}(T_{\text{real},\mathcal{A}'}) \leq \text{Len}_i^r(T_{\text{real},\mathcal{A}'}) + c]\end{aligned}$$

where we also used that for $t = 0$, the length distributions induced by $\mathcal{A}$ and $\mathcal{A}'$ are identical. Hence, chain growth can be analyzed w.r.t. adversary $\mathcal{A}'$ to yield a useful statement for any adversary $\mathcal{A}$.

Let us use the following terminology: We say a round r' is *uniform* if $\text{Len}_i^{r'}(tr) = \text{Len}_j^{r'}(tr)$ holds (where tr is a transcript), for all honest-and-synchronized miners i and j. Recall that adversary $\mathcal{A}'$ does not broadcast adversarially generated states and any new state is delayed by exactly Δ rounds. The slowest progress of the overall maximal state length known to an honest-and-synchronized party occurs in case uniform rounds are the only successful rounds (if at all). Otherwise, the honest miner with the longest state could be successful and broadcast a longer state at round r', which would be guaranteed to arrive to any other honest miner in $r + \Delta$. Furthermore, by a standard coupling argument, the probability of success of any honest-and-synchronized party in some round r' is minimized by an environment $\mathcal{Z}$ that activates just enough parties to obey the mining pattern $\alpha_{r'}$. The coupling with any other environment can be obtained by letting the activation results be the same up to the point where enough parties have been activated to satisfy the mining pattern. Further activations honest-and-synchronized participants can only induce more successful state extension than what $\mathcal{Z}$ obtained.

We are thus left with analyzing growth w.r.t. a simple adversary and an environment $\mathcal{Z}$ with a fixed activation pattern per round to match the mining pattern.

Obtaining a tail bound depending on number of blocks. Now, fix some round r. If in round $s = r + t$, the length increase of the overall longest state of an honest-and-synchronized miner is less than c blocks, then at most $c \cdot \Delta$ non-uniform rounds occurred. According to above,

we can associate to each round i a random variable X_i which is 1 if at least one honest-and-synchronized miner successfully extended the state by a query to $\mathcal{F}_{\text{STX}}$. The X_i's are independent by construction and there must be at least $t - c \cdot \Delta$ uniform rounds. On the other hand, for any concrete sub-sequence of rounds $S \subset (r, \ldots, r+t)$ of size t', the Chernoff-Hoeffding bound in Theorem 2.5.1 implies for our setting (of independent heterogeneous variables) that

$$\Pr\left[\sum_{i \in S} X_i \leq (1-\delta) \cdot \overline{\alpha}_S \cdot t'\right] \leq \exp(-\Omega(\overline{\alpha}_S \cdot t')), \tag{8.2}$$

where $\overline{\alpha}_S := \frac{1}{t'} \sum_{i \in S} \alpha_i$.

We conclude that if for the sub-sequence S of rounds in the interval from r to s, the relations $c = \mathbb{E}\left[\sum_{i \in S} X_i\right] = \overline{\alpha}_S \cdot t'$ and $|S| =: t' = t - c\Delta$ hold, we can derive a tail-estimate depending on the number of blocks. We can define

$$c_S := \frac{\overline{\alpha}_S t}{1 + \overline{\alpha}_S \Delta}$$

and assign a corresponding growth coefficient

$$\gamma_S := \frac{\overline{\alpha}_S}{1 + \overline{\alpha}_S \Delta}.$$

and thus except with exponentially small probability in $t\gamma_S = c_S$, the length-increase is at least c_S for this particular interval.

For the first part of the statement, observe that $\overline{\alpha}_S \geq \alpha_{min}$, for all subsets S, and that the function $\frac{x}{1+kx}$, where k is a positive integer and $x \in (0,1)$, is monotone in x. We get the guaranteed minimal growth by $t \cdot \gamma_{min}$ in any interval of size t rounds for an honest-and-synchronized party except with negligible probability in $t \cdot \gamma_{min}$ by taking the union bound overall all rounds r. What remains to prove is that this bound applies also to the growth of the state if one compares any two honest-and-synchronized miners which we do below (still following the proof steps of [PSS17]).

For the second part of the statement, we generalize the above observation: if we have a guaranteed lower bound τ on the average $\overline{\alpha}_S$ (better than α_{min} as used before) with respect to *any* subset of the required size within

the given interval $r, \ldots, r+t$ (note that indeed we only have a bound for the size of S in our experiments but no guarantee that a particularly "good" one is chosen), the second part of the statement follows.

Bound for any honest-and-synchronized party. By Lemma 8.6.10, we know that if an honest-and-synchronized miner knows some state, then within Δ rounds, every other honest miner will be aware of that state. A similar calculation shows that the lower bound on the time to have a state increase by T blocks by all honest-and-synchronized parties follows the same law (and hence the perceived ledger speed is the same). By requiring $s = r + t - \Delta$ above, and thus considering $t' := t - \Delta - c \cdot \Delta = t - (c+1)\Delta$ does not change the asymptotic behavior since $\gamma_S t - 1 < \gamma_S t - \gamma_S \Delta < \gamma_S t$ for all t and S since $\Delta\gamma_S < 1$. Hence, this additional additive term can be compensated by choosing a sufficiently small constant δ in equation (8.2). □

Mining limits. We state some helpful facts about bounds on the mining behavior.

Lemma 8.6.12. *The number of successful state-extensions that happen with $\mathcal{F}_{\text{StX}}^{\Delta,p_H,p_A}$ in any given interval of t rounds (in the real-world execution under the theorem conditions), where $p_A = p$ and $p_H = 1-(1-p)^q$ for some $q \geq 1$ and $p \in (0,1)$ is bounded by $(1+\delta) \cdot t \cdot \mathtt{T}_{mp}$ for any $\delta > 0$, except with probability $\mathit{negl}(\mathtt{T}_{mp} \cdot t)$. Consequently, for a number T of state-extensions to occur, the number of required rounds is less than $\frac{T}{(1+\delta)\mathtt{T}_{mp}}$ only with negligible probability in T. Finally, the number of adversarial state extensions in a sub-set S of t rounds is no more than $(1+\delta)\overline{\beta}_S \cdot t$ except with probability $\exp(-\Omega(\overline{\beta}_S \cdot t))$ (for any $\delta > 0$.*

Proof. Since the state-exchange functionality evaluates each query independently, we can upper bound the number of successes of these independent Bernoulli-trials. We prove the bound for the environment $\mathcal{Z}$ (and $\mathcal{A}$) that makes as many queries as allowed per round (as limited by β_r and $\mathtt{T}_{mp}$). As in the previous lemma, a coupling argument shows that any other query-distribution cannot induce a larger probability exceeding the given bound than $\mathcal{Z}$, for which the query distribution is fixed. For a round, let $X^{(r)} = \sum_i X_i$ model the sum of the involved independent trials to the state-exchange functionality. Clearly, $\beta_r \leq \mathbb{E}[X^{(r)}] \leq \mathtt{T}_{mp}$.

Let S be a set of t rounds. By linearity of expectation and invoking Theorem 2.5.1 we get the tail-estimate

$$\Pr\left[\sum_{i\in S} X^{(i)} \geq (1+\delta)\cdot t\cdot \mathtt{T}_{mp}\right] \leq \exp(-\Omega(\overline{\beta}_S \cdot t)) \leq \exp(-\Omega(\mathtt{T}_{mp}\cdot t)),$$

where the last step invokes the theorem assumption that $\forall r : \beta_r \geq \rho_a \mathtt{T}_{mp}$ for the relative-power coefficient ρ_a.

Similarly, denote by $Y^{(r)} = \sum_i Y_i$ the number of adversarial state-extensions in round r. Again it is sufficient to consider a maximizing $\mathcal{Z}$ which has an expected value of $t\cdot\overline{\beta}_S$ over a sub-set of rounds of size t. Hence, we again can obtain an estimate of the form

$$\Pr\left[\sum_{i\in S} Y^{(i)} \geq (1+\delta)\cdot t\cdot \overline{\beta}_S\right] \leq \exp(-\Omega(\overline{\beta}_S\cdot t)).$$

As a final conclusion we observe that for any number of state blocks T, the probability that for any $\delta > 0$ it takes less than $t = \frac{T}{(1+\delta)\mathtt{T}_{mp}}$ rounds to get T state extensions is negligible in T. Consequently, for this large time interval, all tail bounds hold except with probability $\exp(-\Omega(T))$, where the constant hidden in $\Omega(\cdot)$ depend on δ and on the relative-power coefficient ρ_a. □

Block withholding. From chain growth and the theorem's condition, we derive that if an honest-and-synchronized miner adopts a new state that contains a block the adversary obtained by $\mathcal{F}_{\text{StX}}$ then either this block has been published by the adversary before, or it was mined quite recently by a corrupted party.

Lemma 8.6.13 (Bound on Withholding strategies)**.** *In the real-world execution (under the conditions of the theorem), assume that in round r, an honest-and-synchronized miner adopts a new longer state* `state`*. Assume there is a block* `st` *in this new state that was accepted upon an adversarial query to $\mathcal{F}_{\text{StX}}$ and that is not part of any state adopted by any honest-and-synchronized party before round r. The probability that such a block* `st` *was first accepted by $\mathcal{F}_{\text{StX}}$ before round $r - \omega t$ happens*

only with probability negl($\overline{\beta}_S \cdot t$), for any constant $0 < \omega < 1$, where S denotes the interval $r - \omega t, \ldots, r$.

Proof. Let us define $\vec{\mathtt{st}}^{(r)} = \mathtt{st}_0 || \ldots || \mathtt{st}_k$ to be the state adopted by the honest-and-synchronized miner in round r as assumed in the lemma statement. Let $\vec{\mathtt{st}}^{(r')}$ be the longest prefix of $\vec{\mathtt{st}}^{(r)}$ such that $\vec{\mathtt{st}}^{(r')}$ is either the genesis block or a state newly accepted by $\mathcal{F}_{\text{LTX}}$ upon a query by an honest-and-synchronized party in round $r' \leq r$. Hence all the blocks in that prefix are known to at least one honest-and-synchronized party by round r'. In light of the lemma statement, we consider the case that $r - r' \geq \omega t$.

Let S denote the set of rounds from r' to r. The number of new states mined by the adversary does not exceed $(1+\delta') \cdot \overline{\beta}_S \omega t$ (except with probability negl($\overline{\beta}_S \cdot t$)) by the previous lemma.

At the same time, equation (8.1) implies that on any subset S' of size $t' = \omega t(1 - \alpha_{max}\Delta)$ the condition $\overline{\alpha}_{S'}(1 - \Delta\overline{\alpha}_{S'}) \geq (1+\delta)\overline{\beta}_S$ has to hold for some constant $\delta \in (0,1)$. This is the case since for all $x, \Delta > 0$, $\frac{x}{1+x\Delta} > x(1 - x\Delta)$ (and $\mathtt{T}_{mp} \geq \overline{\alpha}_{S'}$) and this implies that $\gamma := \frac{\overline{\alpha}_{S'}}{1+\overline{\alpha}_{S'}\Delta} \geq (1+\delta)\overline{\beta}_S$. Lemma 8.6.11 gives us a chain-growth of $|\vec{\mathtt{st}}^{(r)}| - |\vec{\mathtt{st}}^{(r')}| \geq (1-\delta') \cdot \gamma\omega t$ except with probability negl($\overline{\beta}_S \cdot t$).

Since all $|\vec{\mathtt{st}}^{(r)}| - |\vec{\mathtt{st}}^{(r')}|$ blocks must have been mined by the adversary we have $|\vec{\mathtt{st}}^{(r)}| - |\vec{\mathtt{st}}^{(r')}| \leq (1+\delta'') \cdot \overline{\beta}_S \omega t$. We get a contradiction, since now

$$(1-\delta') \cdot \gamma\omega t \leq (1+\delta'') \cdot \overline{\beta}_S \cdot \omega t,$$

which, for sufficiently small δ', δ'' would imply that $\gamma < (1+\delta)\overline{\beta}_S$. □

Chain-growth upper-bound. Our ledger also restricts the growth over time. This is based on the following observation.

Lemma 8.6.14 (Chain-Growth Upperbound)**.** *Consider the real-world execution (under the conditions of the theorem) and let P_i be a miner, and let $r \geq 0$. Assume P_i is honest-and-synchronized in round r, and the longest state received or stored by P_i in round r has length ℓ. Then, in round $r+t$, it holds, except with probability $R \cdot$negl(T), that the length of the longest state (received or stored) of at least one honest-and-synchronized*

miner P_j in that round has length at most $\ell + T$ if $t \leq \frac{T}{(1+\delta)\cdot \mathtt{T}_{mp}}$ for any $\delta > 0$.

Proof. We can combine the previous observations to upper bound the number of accepted blocks. By Lemma 8.6.12 the number of rounds to generate T new extensions of states is at least $t' \geq \frac{T}{(1+\delta')\mathtt{T}_{mp}}$ except with probability $\mathrm{negl}(T)$ (for any $\delta' > 0$) and thus with overwhelming probability, in $t' \leq \frac{T}{(1+\delta')\mathtt{T}_{mp}}$, no more than T new blocks are mined.

In addition, we can invoke Lemma 8.6.13 to conclude that a new state that contains a block that the adversary is withholding since a round prior to $r - \omega t$ is accepted by an honest-and-synchronized party only with probability $\mathrm{negl}(\beta_{min} t)$, for any $0 < \omega < 1$ (since β_{min} can be achieved in any round by an adversarial strategy and hence can serve as the lower bound in the exponent of the tail bound). Analogously to Lemma 8.6.12, by the definition $\rho_a \cdot \mathtt{T}_{mp} = \beta_{min}$ this error probability is thus negligible in T.

Both observations together imply that in $t' = t(1+\omega) \leq \frac{T}{(1+\delta')\mathtt{T}_{mp}}$ rounds, no honest-and-synchronized party experiences a state increase of more than T blocks for any δ' except with negligible probability in T. This is equivalent to the condition that $t \leq \frac{T}{(1+\omega)(1+\delta')\mathtt{T}_{mp}}$ and we can choose δ' sufficiently small to obtain the bound with respect to $t \leq \frac{T}{(1+\delta)\mathtt{T}_{mp}}$ and any given $\delta > 0$ as required by the statement. The claim follows by taking the union bound over all rounds as the arguments above hold for any round r. □

Worst-case chain quality. We give a very coarse bound on the overall chain quality in any sequence of T blocks as follows:

Lemma 8.6.15 (Fraction of honest blocks)**.** *Let P_i be a miner, and let $r \geq 0$. Assume P_i is honest-and-synchronized in round r and that the length of the longest state received or stored is $\ell \geq T$. The fraction of adversarially mined blocks within a sequence of T blocks in the state is at most $\min\{1, (1+\delta)\cdot \frac{\beta_{max}}{\gamma_{min}}\}$ except with probability $R \cdot \mathrm{negl}(T)$ for any $\delta > 0$.*

Proof. Let us assume that at round r, the state adopted by miner P_i is $\vec{\mathtt{st}}_{r'} = \mathtt{st}_0 || \ldots || \mathtt{st}_k$. We show that in any sub-sequence of T state blocks $\mathtt{st}_{j+1}, \ldots, \mathtt{st}_{j+T}$ in $\vec{\mathtt{st}}_r$, the fraction of adversarially mined blocks

is bounded. Without loss of generality, one can assume that the state $\vec{\mathtt{st}}^{<j} := \mathtt{st}_0||\ldots||\mathtt{st}_j$ as well as the state $\vec{\mathtt{st}}^{>j+T} := \mathtt{st}_0||\ldots||\mathtt{st}_{j+T+1}$ are mined by honest-and-synchronized miners (or $j+T$ equals the length of the state). Otherwise, one can enlarge T to meet this condition — this can only increase the fraction of adversarial blocks in the sequence of T blocks and since any state is finite and starts with the genesis block, the condition will be fulfilled for some T. We further assume that $\vec{\mathtt{st}}^{<j}$ is mined at round r', and that in round $r'+t$, the state $\vec{\mathtt{st}}^{>j+T}$ appears for the first time as the state, or the prefix of a state, of at least one honest-and-synchronized miner. We conclude that if an adversary successfully extended the state during some round by a new state block $\mathtt{st}_{j+s}$ of the above sequence $\mathtt{st}_{j+1},\ldots,\mathtt{st}_{j+T}$, then this happens in a round between r' and $r'+t$.

We now relate the number t of rounds to the number T of blocks. Since t is assumed to be the minimal number of rounds until the first honest-and-synchronized miner adopted a state containing $\mathtt{st}_{j+1}$, we can make use of the minimal chain-growth Lemma 8.6.11 to conclude that the probability that the condition $t > \frac{T}{(1-\delta')\gamma_{min}}$ occurs in such an execution is at most $\mathrm{negl}(T)$. We hence have $t \leq \frac{T}{(1-\delta')\gamma_{min}}$ with overwhelming probability in T.

Similar to above, by Lemma 8.6.12 the time it takes to generate T blocks is at least $t \geq \frac{T}{(1+\delta)\mathtt{T}_{mp}}$ except with probability $\mathrm{negl}(T)$ and thus with overwhelming probability, in $t \leq \frac{T}{(1+\delta)\mathtt{T}_{mp}}$, no more than T blocks are mined.

Furthermore, also by Lemma 8.6.12, we get a worst-case upper bound. Let N_A^t denote the expected value in t rounds, invoking Lemma 8.6.12 gives us that $N_A^t \leq (1+\delta)\beta_{max}t$ except with probability $\mathrm{negl}(\beta_{min}t)$ (where we again use the minimum to bound the average of any interval). Hence, since $\rho_a \cdot \mathtt{T}_{mp} = \beta_{min}$ by definition it follows as in previous lemmata that the bound holds except with probability $\mathrm{negl}(T)$.

Putting things together, we conclude that except with negligible probability in T, the number of times the adversary was successful in extending the state by one block is upper bounded by the quantity

$$N_A^{\frac{T}{(1-\delta')\gamma}} \leq \frac{1+\delta}{1-\delta'} \cdot T \cdot \frac{\beta_{max}}{\gamma_{min}}.$$

Hence, the fraction of adversarial blocks within T consecutive blocks

cannot be more than $f = \min\{1, (1+\delta'')\frac{\beta_{max}}{\gamma_{min}}\}$ for any δ'' and sufficiently small constants $\delta, \delta' > 0$, except with negligible probability in the length T of the sequence.

Since our arguments hold for any interval, the proof is concluded by taking the union bound over the number of such sequences (which is in the order of number of rounds). □

Consistency (common prefix). We now state the lemma on the common-prefix property in our setting.

Lemma 8.6.16 (Consistent states)**.** *Consider the real-world execution under the condition of the main theorem. Let P_i and P_j be miners (potentially the same), and let $r' \geq r \geq 0$. Assume P_i is honest-and-synchronized in round r, and P_j is honest-and-synchronized in round r'. Assume that the length of the longest state received or stored by P_i in round r is $\ell \geq T$. Then, the $\ell - T$-prefix of that longest state of P_i in round r is identical to the $\ell - T$-prefix of the state of P_j stored or received in round r' except with probability $R \cdot negl(T)$.*

Proof. We again follow the basic line of reasoning in [PSS17] and adapt the appropriate arguments to our setting. First, since an inconsistency at round r implies an inconsistency at round $r' > r$, if the claim is proven for the case $r \leq r' \leq r + 1$, then by an inductive argument, the claim holds for any $r' \geq r$.

The protocol mandates that the honest-and-synchronized miners truncates the T newest blocks from the current respective state. Thus, we need to argue that the block which is $T+1$ far away from the head will be part of any state output by any honest-and-synchronized miner. Suppose we are at round r' in the protocol, then the time it takes to generate the last T blocks is at least $t \geq \frac{T}{(1+\delta)\mathtt{T}_{mp}}$ except with negligible probability in T as established in Lemma 8.6.12 and any $0 < \delta < 1$.

Looking ahead, we will eventually conclude that with overwhelming probability within the interval of rounds $s = r - t, \ldots, r' \in \{r, r+1\}$ (where $r \geq t$), the honest-and-synchronized miners have an opportunity to agree on a common state and hence at round r', they will still agree on a large common prefix of the current state at round r'.

In the interval of rounds, let this set be denoted as usual by S, between round s and round $r' = r$, the expected number of rounds, where at lest

one honest-and-synchronized miner is successful, is at least $\overline{\alpha}_S t$. Thus, again by a standard Chernoff bound, the probability that the number of these successful rounds is smaller than $\bar{q}_{min} := (1-\delta') \cdot \overline{\alpha}_S t$ is no more than $\exp(-\Omega(t\overline{\alpha}_S))$ in the real-world UC random experiment. Again, a coupling argument as in Lemma 8.6.11 yields that this tail-bound (where the environment activates the least number of parties possible and hence the random variables that describe the success are independent) applies to any environment. Finally, the conditions of the theorem in particular assure that $\overline{\alpha}_S > \beta_{min}$ and hence this probability can be upper bounded by $\mathrm{negl}(\beta_{min} t)$.

Unfortunately, the "race" between the good guys and the bad guys is not yet conclusively analyzed, since the mere superiority of honestly mined blocks does not imply that the honest parties will reach agreement. In particular, not all of the expected honestly mined blocks are equally useful to obtain a so-called convergence opportunity. In particular, we need to know how many of the honestly mined blocks happen in isolated, sufficiently silent intervals.

Formally, let us introduce the random variable R_i that measures the number of elapsed round between successful round $i-1$ and successful round i in the real-world UC execution, where R_1 measures the number of elapsed rounds to the first successful round. Based on R_i, the random variable X_i is defined as follows: $X_i = 1$ if and only if $R_i > \Delta$ and exactly one honest-and-synchronized miner mines a new state (i.e., successfully appends a new block to the state) in the ith successful round.

Let E_1^i be the event that there is at least one successful round in the interval of Δ rounds starting after successful round $i-1$ (or at the onset of the experiment). Let E_2^i be the event that strictly more than one miner is successful in the following successful round i.

Overall, our goal is to suitably bound the number of blocks that prevent those events of "success & silence" (i.e., bound the probability of the event $X_i = 0$) in an interval of t rounds. We call these the undesirable blocks. They have to be infrequent enough such that in combination with adversarially mined blocks, they do not prevent too many convergence opportunities. We hence need to suitably bound the occurrence of the above two bad events E_j^i in our experiment.

By a union bound, and invoking that $\alpha_r \leq \mathtt{T}_{mp}$, we directly have that $\Pr[X_i = 0] = \Pr[E_1^i \cup E_2^i] \leq \Delta \mathtt{T}_{mp} + \mathtt{T}_{mp}$, hence, on the positive side, $\Pr[X_i = 1] \geq 1 - \mathtt{T}_{mp}(\Delta + 1)$.

Let $X := \sum_{i=1}^{\bar{q}_{min}} X_i$ and let us define $\bar{q}'_{min} := (1-\delta'') \cdot (1 - \mathtt{T}_{mp}(\Delta + 1)) \cdot \bar{q}_{min}$. Since by equation (8.1) the term $1 - 2(\Delta+1)\mathtt{T}_{mp}$ must be positive, we have that $\mathtt{T}_{mp}(\Delta+1) \leq \frac{1}{2}$ and, because $\mathcal{F}_{\text{StX}}$ treats each new state-submission independently of previous submission, we conclude that $\Pr[X_i = 1 \,|\, X_1, \ldots, X_{i-1}] \geq \frac{1}{2}$. Since we do not argue here about any particular optimal strategy by an environment-adversary pair $(\mathcal{Z}, \mathcal{A})$, we need to invoke Lemma 8.6.17 from which we get

$$\Pr[X \leq \bar{q}'_{min}] \leq \exp\left(-(\delta'')^2 \bar{q}_{min}/2\right). \tag{8.3}$$

To express this w.r.t. β_{min}, observe that not only $\alpha_r > \beta_r$ (and thus $\alpha_{min} > \beta_{min}$) by equation (8.1) but also there is an actual constant $0 < \widehat{\delta} < 1$ such that $\mathtt{T}_{mp}(\Delta+1) < 1 - \widehat{\delta}$. This is true since by the theorem condition we deduce that

$$\begin{aligned} &(1 - 2(\Delta+1)\mathtt{T}_{mp}) \geq \lambda(\beta_{min}/\alpha_{min}) \\ &\implies 1 - \lambda(\beta_{min}/\alpha_{min}) \geq 2(\Delta+1)\mathtt{T}_{mp} > (\Delta+1)\mathtt{T}_{mp}. \end{aligned}$$

And since $\lambda > 1$, i.e., we get can bound the constant by $0 < \widehat{\delta} < \lambda(\beta_{min}/\alpha_{min})$ and obtain

$$(1 - \mathtt{T}_{mp}(\Delta+1)) \cdot \bar{q}_{min} > \widehat{\delta}(1-\delta') \cdot \overline{\alpha}_S t > \widehat{\delta}(1-\delta') \cdot \overline{\beta_S} t.$$

And hence conclude by equation (8.3) that $\Pr[X \leq \bar{q}'_{min}] \leq \exp(-\Omega(\beta_{min} t))$. We thus have a (high-probability) lower bound on the number of silent patterns.

We are actually interested in the number of times we have $X_i = X_{i+1} = 1$. This situation, as introduced above, means that we have a situation, in which for Δ rounds, no miner is successful, then exactly one honest-and-synchronized miner is successful, and afterwards, we again have Δ rounds of silence. This is denoted in [PSS17] as a *convergence opportunity*. For example, a convergence opportunity has the desirable property, that *at the end* of such an opportunity, if the adversary is unable to provide a longer state to the honest-and-synchronized miners during this period, all honest-and-synchronized miners will reach an agreement on the current longest state. Thus, in order to prevent this, an adversary needs to be successful in mining roughly at the rate of the number of convergence opportunities within t rounds.

We have already seen that with overwhelming probability, there are at least $\bar{q}_{min}$ successful rounds, and among which $(\bar{q}_{min} - \bar{q}'_{min})$ can disturb convergence opportunities. Since a single disturbing round can at most prevent two convergence opportunities (it violates the condition for a convergence opportunity with its neighbors in the sequence $X_1, \ldots, X_k$), the number of effective convergence opportunities, say C, is lower bounded (except with neglgiblble probability) by

$$\begin{aligned} C &\geq \bar{q}_{min} - 2(\bar{q}_{min} - \bar{q}'_{min}) = 2\bar{q}'_{min} - \bar{q}_{min} \\ &\geq (1-\delta')\overline{\alpha}_S t[1 - 2\mathtt{T}_{mp}(\Delta+1) - 2\delta'']. \end{aligned}$$

For any constant ϵ, by picking δ' and δ'' sufficiently small, this yields a bound (except with negligible probability as derived above) of

$$C > (1-\epsilon)(1 - 2\mathtt{T}_{mp}(\Delta+1))\overline{\alpha}_S t.$$

The final argument is a counting argument. Let us denote by $\mathcal{S}_{r'}$ the set of maximal states known to $\mathcal{F}_{\text{StTX}}$ at round r' (i.e., any path from the root to some a leaf) of length at least $\ell + C$, where ℓ is the length of the longest state known to at least one honest-and-synchronized miner at round s. Note that $\mathcal{S}_{r'}^{\ell+C}$ is non-empty: since each convergence opportunity increases the length by at least one, and before each successful round, there is a period of Δ rounds where no honest miner mines a new state, there has to exist at least one state with length at least $\ell + C$ at round r'.

Assume that the number of successful state extensions made by the adversary (to $\mathcal{F}_{\text{StTX}}$) between round s and r' is $T_{\mathcal{A}} < C$. Then, by the pigeonhole principle, for all $\vec{\mathtt{st}} \in \mathcal{S}_{r'}$, it holds that there is at least one block $\mathtt{st}_k$, such that functionality $\mathcal{F}_{\text{StTX}}$ is successfully queried by exactly one honest-and-synchronized miner P in round i to extend the state to length $k+1$, but no query by the adversary to extend a state of length k to a state of length $k+1$ has been successful up to and including round r'. Even more, $T_{\mathcal{A}} < C$ implies that such an i has to exist that also constitutes a convergence opportunity.

After this convergence opportunity at round i, all honest-and-synchronized miners have a state whose first $k+1$ blocks are $\vec{\mathtt{st}}_i = \mathtt{st}_0 \ldots, \mathtt{st}_k$. Unless the adversary provides an alternative state with a prefix $\vec{\mathtt{st}}'_i$ of length $k+1$, such that $\mathtt{st}'_l \neq \mathtt{st}_l$ for at least one index $0 < l \leq k$, no honest-and-synchronized miner will ever mine on a state whose first $k+1$ blocks do not agree with $\vec{\mathtt{st}}_i$.

The existence of an alternative prefix $\vec{\mathtt{st}}'_i$ of length $k+1$ for any such i and for all states $\vec{\mathtt{st}} \in \mathcal{S}^{\ell+C}_{r'}$ implies $T_{\mathcal{A}} \geq C$ and therefore contradicts the assumption that $T_{\mathcal{A}} < C$.

What is left to prove is that for any such interval of size t (from round s to round r'), the probability that $T_{\mathcal{A}} < C$ holds in any real-world execution except with negligible probability in $\beta_{min}t$. Analogously to Lemma 8.6.12, by the definition $\rho_a \cdot \mathtt{T}_{mp} = \beta_{min}$ (and recalling that we established a lower bound on t in the beginning) we get that this error probability is negligible in T.

First, by Lemma 8.6.13, for any $\omega > 0$, the probability that any new state accepted by an honest-and-synchronized miner during the period of at most $t+1$ rounds (from s to r') is actually a state extension that the adversary withheld since round $s - \omega(t+1)$ (or even before) is at most $\mathrm{negl}(\beta_{min}t)$. By Lemma 8.6.12, the number of adversarial blocks (i.e., successful state extensions by $\mathcal{A}$) generated within this slightly larger interval S' of size $|S'| = (1+\omega)(t+1)$ rounds is (except with probability $\mathrm{negl}(\beta_{min}t)$) upper bounded by $T_{\mathcal{A}} \leq (1+\delta)(1+\omega)(t+1)\overline{\beta}_{S'}$. Also by picking constant ω sufficiently small, we have that $|S| \geq (1-\alpha_{max}\Delta)|S'|$ and thus $\overline{\alpha}_S$ dominates $\overline{\beta}_{S'}$ by the theorem assumptions. We hence get $T_{\mathcal{A}} \leq \frac{(1+\delta)(1+\omega)}{\lambda}(t+1)\overline{\alpha}_S \cdot (1 - 2\mathtt{T}_{mp} \cdot (\Delta+1))$ by equation (8.1). By picking the constants δ and ω, and ϵ sufficiently small relative to λ, we hence get $T_{\mathcal{A}} < C$ except with probability $\mathrm{negl}(\beta_{min}t)$. Since our arguments hold for any particular intervals S, we again apply the union bound over the number of rounds and get the desired claim. □

We used the following useful lemma in the previous proof to bound the deviation with respect to an arbitrary environment (inducing a certain sequence of random variables):

Lemma 8.6.17. *Let $\tau \geq \frac{1}{2}$ and consider boolean random variables $X_1, \ldots, X_n$ for which it holds that $\Pr[X_i = 1 \mid X_1, \ldots, X_{i-1}] \geq \tau$. Then, for any $\delta > 0$,*

$$\Pr[\sum_{i=1}^{n} X_i \leq (1-\delta)\tau n] \leq \exp\left(-\delta^2 n/2\right).$$

Proof. We define the random variables $Y_k := \sum_{i=1}^{k}(X_i - \tau) = (\sum_{i=1}^{k} X_i) - k\tau$. First, they satisfy the sub-martingale condition, i.e., for all k,

$\mathbb{E}[Y_k \,|\, Y_1, \ldots Y_{k-1}] \geq Y_{k-1}$: let $\Pr[Y_k = y_{k-1} + (1-\tau) \,|\, Y_{k-1} = y_{k-1}] = \Pr[X_k = 1 \,|\, X_1, \ldots, X_{k-1}] =: p_1 \geq \tau$ and $\Pr[Y_k = y_{k-1} + (-\tau) \,|\, Y_{k-1} = y_{k-1}] = \Pr[X_k = 0 \,|\, X_1, \ldots, X_{k-1}] := p_0 \leq 1 - \tau$. The (conditional) expected value is $p_1(y_{k-1} + (1-\tau)) + p_0(y_{k-1} - \tau) \geq y_{k-1} + p_1(1-\tau) - p_0\tau \geq y_{k-1} + [\tau(1-\tau) - (1-\tau)\tau] = y_{k-1}$. Second, we have a bounded difference of $|Y_k - Y_{k-1}| \leq \max(\tau, 1-\tau) = \tau$ by the condition $\tau \geq 1/2$. Applying the Azuma-Hoeffding bound given by Theorem 2.5.2 to the variables Y_k gives

$$\Pr[Y_n \leq -\delta\tau n] \leq \exp(-\delta^2 n/2).$$

And by definition $Y_n \leq -\delta\tau n \leftrightarrow X_n \leq n\tau - n\delta\tau$, the statement follows. □

Concluding observations. Finally, we conclude the proof by noting that after a delay of Δ rounds, all honestly multicast transactions are known to all honest-and-synchronized miners and would be included into the next honestly minded block if valid. In the simulation, the simulator also does it in the ideal world and hence will never propose blocks of honest parties that do not comply with the conditions of the defined $\mathsf{ExtendPolicy}$ of $\mathcal{G}^{\mathcal{B}}_{\text{LEDGER}}$. Further, the synchronization of a party takes at most $\texttt{Delay} = 4\Delta$ clock ticks: if P_j joins the network, his knowledge of the longest chain and the set of valid transactions relative to that state, which is known to at least one honest and synchronized miner is only reliable after 2Δ rounds (4Δ clock ticks) since it takes at most Δ rounds to multicast the initial message that the miner has joined the network, and additional Δ rounds until the replies are received. During this 2Δ round the new miner will also have received all messages sent at or after he joined the network, and in particular all transactions that are more than Δ rounds ($2\Delta = \frac{\texttt{Delay}}{2}$) old and potentially valid.

The pointers of honest-and-synchronized parties can also not be too distant, i.e., the slackness is upper bounded by $\texttt{windowSize} \geq T$ as otherwise we would have a common-prefix violation in that execution (assume the prefix of the chain known to a honest-and-synchronized party was further away than T blocks from the prefix of the actual longest chain, this would yield a fork with substantial probability). The theorem follows. □

8.6.4 Improving the Chain-Quality Parameter

As long as $\alpha_{min} > \beta_{max}$, we see that among `windowSize` state blocks, there is at least an honestly generated block, because then, by equation (8.1), we also have $\gamma_{min} > \beta_{max}$ and thus $\frac{\beta_{max}}{\gamma_{min}} < 1$. Such an assumption is usually taken in existing analyses. However, we can derive more general bounds for chain-quality (where the above case is one special case) to obtain bounds for more general scenarios. In light of the chain-growth statement in Lemma 8.6.11, we introduce the following useful quantity:

Definition 8.6.18. Let the mining pattern be $(\vec{\alpha}, \vec{\beta})$ for R rounds, let the network delay be Δ, and let S be an interval. Define

$$cg_\Delta(S) := \max\{\tau \in (0,1) \,|\, \forall S' \subseteq S \text{ with } |S'| \geq \max\{1, |S|(1-\Delta\cdot\gamma(\tau,\Delta))\} : \overline{\alpha}_{S'} \geq \tau\};$$

and define the fraction

$$f_{cq} := \max_{S \subseteq \{0,\ldots,R-1\}} \frac{\overline{\beta}_S}{\gamma(cg_\Delta(S),\Delta)}.$$

Both quantities are well-defined as functions since we assume that $\forall r : \alpha_r > 0$. We derive a more general worst-case guarantee for the fraction of adversarial blocks and we see that the fraction of adversarial blocks is actually less than one under the theorem condition.

Lemma 8.6.19 (Generalization of Lemma 8.6.15)**.** *Consider a real-world execution as in Theorem 8.6.9. Let P_i be a miner, and let $r \geq 0$. Assume P_i is honest-and-synchronized in round r and that the length of the longest state received or stored is $\ell \geq T$. The fraction of adversarially mined blocks within a sequence of T blocks in the state is at most* $\min\{1, (1+\delta)\cdot f_{cq}\}$ *except with probability $R \cdot negl(T)$ for any $\delta > 0$ and where f_{cq} is defined as in Definition 8.6.18. Under the condition of Theorem 8.6.9, this means that for the ledger $\mathcal{G}^{\hat{p}}_{\text{LEDGER}}$, we can guarantee*

$$\eta \geq \min\{(1+\delta)\cdot f_{cq} \cdot \texttt{windowSize}, \texttt{windowSize}\},$$

with $f_{cq} < 1$ (and for any $\delta > 0$).

Proof. The proof proceeds as the one of Lemma 8.6.15: consider any subsequence of T state blocks $\mathtt{st}_{j+1}, \ldots, \mathtt{st}_{j+T}$ in $\vec{\mathtt{st}}_r$. We again assume that $\vec{\mathtt{st}}^{<j}$ is mined at round r' (by an honest-and-synchronized party), and that in round $r' + t$, the state $\vec{\mathtt{st}}^{>j+T}$ appears for the first time as the state, or the prefix of a state, of at least one honest-and-synchronized miner. Recall that if an adversary successfully extended the state during some round by a new state block $\mathtt{st}_{j+s}$ of the above sequence $\mathtt{st}_{j+1}, \ldots, \mathtt{st}_{j+T}$, then this happens in a round between r' and $r' + t$. Let us denote this interval by the set S of rounds.

Since t is assumed to be the minimal number of rounds until the first honest miner adopted a state containing $\mathtt{st}_{j+1}$, we can actually make use of the general part of Lemma 8.6.11 to conclude that the probability that the condition $t \geq \frac{T}{(1-\delta')\gamma(cg_\Delta(S),\Delta)}$ occurs in such an execution is at most $\mathrm{negl}(T)$ and obtain $t \leq \frac{T}{(1-\delta')\gamma(cg_\Delta(S),\Delta)}$ with overwhelming probability in T. On the other hand, the lower bound on t is as in the proof of Lemma 8.6.15.

Let again N_A^t denote the expected value of adversarial blocks in t rounds, invoking Lemma 8.6.12 gives us that $N_A^t \leq (1+\delta)\overline{\beta}_S t$ except with probability $\mathrm{negl}(\overline{\beta}_S t)$.

The number of times the adversary was successful in extending the state by one block can therefore be upper bounded by the quantity

$$N_A^{\frac{T}{(1-\delta')\gamma}} \leq \frac{1+\delta}{1-\delta'} \cdot T \cdot \frac{\overline{\beta}_S}{\gamma(cg_\Delta(S),\Delta)}.$$

Since our arguments hold for any interval, the proof is concluded by taking the worst case over all rounds and the maximal fraction equals f_{cq} as claimed.

To establish the last part of the statement, we observe that equation (8.1) in particular implies that for any interval S (of sufficient size), we have that any subset S' of rounds of size $(1-\alpha_{max}\Delta)|S|$ fulfills $\overline{\alpha}_{S'}(1-\mathtt{T}_{mp}\Delta) > (1+\epsilon)\overline{\beta}_S$ for some $\epsilon > 0$. Since a lower bound x for $\overline{\alpha}_{S'}$ over all subsets of size $(1-\alpha_{max}\Delta)|S|$ implies that x is also a lower bound for any larger subset S'' and hence for $cg_\Delta(S)$. Observing that for $x, \Delta > 0$, $\frac{x}{1+x\Delta} > x(1-x\Delta)$ and $\mathtt{T}_{mp} \geq cg_\Delta(S)$, we get $\gamma(cg_\Delta(S),\Delta) > \overline{\beta}_S$ as required to conclude that $f_{cq} < 1$. □

8.7 Special Cases of our Model and Functionality Wrappers

In this section, we first explain important special cases of our main theorem and show how to use functionality wrappers to enforce its conditions to obtain composable statements.

8.7.1 Special Cases and Existing Works

We demonstrate how the protocols, assumptions, and results from the two existing works analyzing security of Bitcoin (in a property based manner) can be cast as special cases of our construction. We focus on the analyses of Pass et al. (PSs for short) and of Garay et al. (GKL for short).

These models assume a number n of participants being active in the protocol execution. All honest parties are assumed to be synchronized (e.g., by special initialization messages by the environment).

GKL analysis (fixed difficulty and delay). We start with the result in [GKL15], in particular with the so-called flat and synchronous model with next-round delivery and a constant number of parties n (i.e., Bitcoin is seen as an n-party MPC protocol).[19] The relevant variables are defined as follows:

- Each party is allowed to perform $q \geq 1$ hash queries. This translates to a success probability of $p_H = 1 - (1-p)^q$ and $p_A = p$, and to a total mining power $\mathtt{T}_{mp}^{\mathrm{GKL}} := p \cdot q \cdot n$.
- The adversary gets (at most) q queries per corrupted party with probability $p_A = p$ (there are no desynchronized parties). If t_r denotes the number of corrupted parties in round r, the expected value would be $t_r \cdot q \cdot p$ and thus we can define the upper bound on the adversarial mining power $\beta_{\max}^{\mathrm{GKL}} = p \cdot q \cdot (\rho \cdot n)$, where ρn is the (assumed) upper bound on the number of miners contributing to the adversarial mining power (independent of r). Since the adversary is free to go to the limit in the model, the mining pattern is also flat: $\vec{\beta} = (\beta_{\max}^{\mathrm{GKL}}, \ldots, \beta_{\max}^{\mathrm{GKL}})$.

[19] In a recent paper, the authors of [GKL15] propose an analysis of Bitcoin for a variable number of parties.

- Each honest and synchronized miner gets exactly one activation per round and has success probability $p_H = 1 - (1-p)^q \in (0,1)$, for some integer $q > 0$ and hence we get a minimal honest mining power of $\alpha_{\min}^{\text{GKL}} = 1-(1-p)^{q(1-\rho)\cdot n}$ (independent of r). Note that since n is assumed to be fixed in their model, $q(1-\rho)\cdot n$ is in fact a lower bound on the honest and synchronized hashing power. Since the model assumes that this lower bound could potentially always be allowed, we again define the flat mining pattern $\vec{\alpha} = (\alpha_{\min}^{\text{GKL}}, \ldots, \alpha_{\min}^{\text{GKL}})$.
- If instant delivery is assumed, this translates to defining $\Delta^{\text{GKL}} := 1$, i.e., guaranteed delivery in the next round.

PSs analysis (fixed difficulty). Similarly, we can instantiate the above values with the assumptions of [PSS17]:

- For each corrupted party, the adversary gets at most one query per round. Each honest miner makes exactly one query per round. In total, there are n parties among which ρn can be corrupted (in any round).
- In the PSs model, $p_H = p_A = p$ and hence $\mathtt{T}_{mp}^{\text{PSs}} = p \cdot n$. With these values we get $\beta_{\max}^{\text{PSs}} = p \cdot (\rho \cdot n)$. Putting things together, we also have $\alpha_{\min}^{\text{PSs}} = 1 - (1-p)^{(1-\rho)\cdot n}$, where $(1-\rho)\cdot n$ is the lower bound on the honest (and hence also synchronized) parties. As before, the mining pattern is flat.
- The delay of the network is upper bounded by a constant Δ^{PSs} (as usual, unknown to the participants).

The security is established with the following lemma:

Lemma 8.7.1. *For the special settings above, if we impose the assumption that*

$$\alpha_{\min}^{\{\text{GKL, PSs}\}} \cdot (1 - 2 \cdot (\Delta^{\{\text{GKL, PSs}\}} + 1) \cdot \alpha_{\min}^{\{\text{GKL, PSs}\}}) \geq \lambda \cdot \beta_{\max}^{\{\text{GKL, PSs}\}} \tag{8.4}$$

then this implies the secure realization of the Bitcoin ledger with the parameters assured by Theorem 8.6.9 for the above choices of values, respectively.

Proof Sketch. The statement of course follows from the arguments given in the respective works [GKL15] and [PSS17] since our execution model in particular allows us to formulate the above assumptions. However, it is instructive to see how the security follows in view of Theorem 8.6.9. In particular, why security follows when replacing the condition in equation (8.1) by equation (8.4). At first sight, the condition is stronger as it implies that the best strategy of the adversary is dominated by the worst strategy of the honest players. However, the discount factor $(1 - 2 \cdot (\Delta^{\{\text{GKL, PSs}\}} + 1) \cdot \alpha_{\min}^{\{\text{GKL, PSs}\}})$ is better than $(1 - 2 \cdot (\Delta^{\{\text{GKL, PSs}\}} + 1) \cdot \mathtt{T}_{mp}^{\text{GKL,PSs}})$. The key observation why equation (8.4) subsumes equation (8.1) in the special cases described above are the following:

- Since the number n of parties is fixed and exactly divided into honest and adversarial, and because the worst-case honest strategy still dominates the adversary's best strategy, we can use to following argument to justify why equation (8.4) is actually sufficient. Still, the best strategy of the adversary is to activate as many corrupted parties, say t, as allowed by the upper bound β_{max}. Since the number of parties is fixed, this implies that at most $n - t$ activations of honest parties remain and by definition $\alpha_{min} = 1 - (1 - p)^(n - t)$ is the matching lower bound. Hence, and in contrast to the more general setting, here the best strategy for corrupted parties induces a concrete strategy for honest parties.[20] A bit more formally, let x denote the number of queries such that $\alpha_{min} = 1 - (1 - p)^x$ holds. Assume in some round r, more honest parties are activated, say q_H^r. By definition, $\beta_{max} \geq p \cdot (n - x)$ and we can formally assign the difference $(q_H^r - x)$ to the adversary's budget (and the condition $\alpha_{min} > \beta_{max}$ is preserved as stated below). First, observe that for integers $x, y > 1$,

$$\begin{aligned}\alpha_r - \alpha &= (1 - (1 - p)^{x+y}) - (1 - (1 - p)^x) = (1 - p)^x - (1 - p)^{x+y} \\ &= (1 - p)^x \cdot (1 - (1 - p)^y) \leq (1 - (1 - p)^y) \leq (1 - (1 - y \cdot p)) \\ &= y \cdot p,\end{aligned}$$

  where the last inequality is a consequence of Bernoulli's inequality.

[20] Note that in a more general setting, this not need to be the case: even if the bound on the adversary is small, by activating a huge fraction of honest parties the consensus of honest parties could still be disturbed and hence our analysis has to consider such "malicious" strategies as well.

The adversary's mining power is thus increased, however not beyond β_{max} since the identity $n - x = (n - q_H^r) + (q_H^r - x)$ is guaranteed because n and x are fixed for the analysis.

- Looking at the proof of Theorem 8.6.9, we see that the quantities $\overline{\alpha}_S$ and $\overline{\beta}_S$ can be identified by α_{min} and β_{max}, respectively, and in addition the relationship $\alpha_{min} > \beta_{max}$ is implied by equation (8.4) (and thus $\overline{\alpha}_S > \overline{\beta}_S$ for any subset S of rounds of any size. With this, all Lemmata in the proof of Theorem 8.6.9 simplify and no further condition in addition to equation (8.4) is needed.

With this in mind, replacing the condition in equation (8.1) by equation (8.4) the proof of Theorem 8.6.9, under the conditions imposed by the above models, yields the statement of the lemma. □

8.7.2 Restrictions and Composition

Note that the theorem statement a-priori holds for any environment (but simply yields a void statement if the conditions are violated). In order to turn this into a composable statement without restrictions, we follow the approach proposed in Section 8.2 and model restrictions in the setup of the protocol via wrapper functionalities. The general conceptual principle behind this is the following: For the hybrid world, that consists of a network $\mathcal{F}_{\text{N-MC}}$, a clock $\mathcal{G}_{\text{CLOCK}}$ and a random oracle $\mathcal{F}_{\text{RO}}$ with output length κ (or alternatively the state-exchange functionality $\mathcal{F}_{\text{StX}}$ instead of the random oracle), define a wrapper functionality $\mathcal{W}$ which enforces a given mining pattern $(\vec{\alpha}, \vec{\beta})$ (and the upper bounds on the mining power). If the conditions of Theorem 8.6.9 are met, then we get a UC-realization statement with respect to all (efficient) environments.

A general wrapper. We define a wrapper along the lines of the basic example in Section 8.2 and we provide the details and the specification of such a general random-oracle wrapper $\mathcal{W}_{\vec{\alpha},\vec{\beta},\mathtt{D}}^{\Delta,\mathtt{T}_{mp}}(\mathcal{F}_{\text{RO}})$ in Figure 8.10. This wrapper slightly changes the synchrony pattern of the real-world execution: since a lower bound on honest mining power is enforced (otherwise, the clock does not go on), we realize the ledger with a slightly different predicate $\mathsf{predict\text{-}time}_{BC}$ to reflect this assumption. It is easy to see that this is a straightforward extension to the derivation in Lemma 8.4.2. We

note that this change to the synchronization pattern just stems from the fact how we implement such restricting assumptions but does not affect other modeling decisions. Recall that this is a major motivation to abstract the time-dependency of the ledger using such an abstract predicate, such that minor details have only local effects.

For this wrapper we have the following desired corollary to Theorem 8.6.9 and Lemma 8.6.2. This statement is guaranteed to compose according to the UC composition theorem.

Corollary 8.7.2. *The protocol* $\mathsf{Ledger\text{-}Protocol}_{q,\mathcal{B},T}$*, which is defined in the* $(\mathcal{G}_{\text{CLOCK}}, \mathcal{F}_{\text{N-MC}}^{\Delta}, \mathcal{W}_{\vec{\alpha},\vec{\beta},p}^{\tau_{mp}}(\mathcal{F}_{\text{RO}}))$*-hybrid world, UC-realizes functionality* $\mathcal{G}_{\text{LEDGER}}^{\mathcal{B}}$ *(for the parameters established by Theorem 8.6.9 and the extended predicate* $\mathsf{predict\text{-}time}_{BC}$ *as described above) if the parameters of the wrapper (and thus formally enforced by the setup-functionality of the protocol), satisfy equation* (8.1)*.*

It is straightforward to design different wrappers capturing a range of assumptions that one might want to make (and which imply the conditions of Theorem 8.6.9), such as an explicit restriction on number of active participants etc. Each of these real-world assumptions might influence the time-progress and hence the predict-time-predicate.

8.8 Modular Constructions based on the Ledger

The ledger functionality can be enhanced in a modular way in various directions. In fact, the presented ledger functionality can be seen as the minimal composable goal of a blockchain protocol. Different blockchain protocols would typically achieve different ledgers, either because they achieve stronger guarantees or offer more capabilities in addition to the basic ones we captured. In this section, we show a straightforward extension.

As already observed in [GKL15], the Bitcoin protocol makes use of digital signatures to protect transactions which allows it to achieve stronger guarantees. Informally, the stronger guarantee ensures that every transaction submitted by an honest miner will eventually make it into the state. Using our terminology, this means that by employing digital signatures, Bitcoin implements a stronger ledger. In this section

Functionality $\mathcal{W}^{\Delta,\lambda,\mathtt{T}_{mp}}_{\vec{\alpha},\vec{\beta},\mathtt{D}}(\mathcal{F}_{\mathrm{RO}})$

The wrapper functionality is parametrized by the mining pattern, the difficulty, and the upper bound $\mathtt{T}_{mp}$ on the total mining power per round (which thereby also implies an upper bound on the total number of RO-queries per round). The wrapper is assumed to be registered with the global clock $\mathcal{G}_{\mathrm{CLOCK}}$. The functionality manages the variable `counter` and is aware of set of registered parties, and the set of corrupted parties.

Initially, $\mathcal{P}' = \emptyset$ and $\mathtt{counter} = 0$, $q_A = 0$ and $q_H = 0$. Define $p := \frac{\mathtt{D}}{2^\kappa}$ (where κ is the output length of the underlying random oracle).

General:

- The wrapper stops the interaction with the adversary as soon as the adversary tries to exceed its allowed budget of hashing power.

Relaying inputs to the random oracle:

- Upon receiving $(\textsc{eval}, \mathrm{sid}, x)$ from $\mathcal{A}$ on behalf of a party P which is corrupted or registered but de-synchronized, then first execute *Round Reset.* Then do the following:

  $q_A \leftarrow q_A + 1$; $\beta^{(\mathtt{counter})} \leftarrow q_A \cdot p$
  if $(q_A + q_H) \cdot p \leq \mathtt{T}_{mp}$ **then**
    if $\beta^{(\mathtt{counter})} \leq \vec{\beta}[\mathtt{counter}]$ **then**
      Forward the request to $\mathcal{F}_{\mathrm{RO}}$ and return to $\mathcal{A}$ whatever $\mathcal{F}_{\mathrm{RO}}$ returns.

- Upon receiving $(\textsc{eval}, \mathrm{sid}, x)$ from an uncorrupted, registered and synchronized party P, then first execute *Round Reset.* Then do the following:

  $q_H \leftarrow q_H + 1$; $\alpha^{(\mathtt{counter})} \leftarrow 1 - (1-p)^{q_H}$
  if $(q_A + q_H) \cdot p \leq \mathtt{T}_{mp}$ **then**
    Forward the request to $\mathcal{F}_{\mathrm{RO}}$ and return to P whatever $\mathcal{F}_{\mathrm{RO}}$ returns.
  if $\alpha_{\mathtt{counter}} \geq \vec{\alpha}[\mathtt{counter}]$ **then**
    Send $(\textsc{clock-update}, \mathrm{sid}_C)$ to $\mathcal{G}_{\mathrm{CLOCK}}$ ▷ Release the clock if lower bound is reached.

- Any other request is relayed to the underlying functionality (and recorded by the wrapper) and the corresponding output is given to the destination specified by the underlying functionality.

Standard UC Corruption Handling:

- Upon receiving $(\textsc{corrupt}, \mathrm{sid}, P)$ from the adversary, set $\mathcal{P}' \leftarrow \mathcal{P}' \cup \{P\}$.

Procedure Round-Reset:
Send $(\textsc{clock-read}, \mathrm{sid}_C)$ to $\mathcal{G}_{\mathrm{CLOCK}}$ and receive $(\textsc{clock-read}, \mathrm{sid}_C, \tau)$ from $\mathcal{G}_{\mathrm{CLOCK}}$. If $|\tau - \mathtt{counter}| > 0$ and the new time τ is even (i.e., a new round started), then set $\mathtt{counter} \leftarrow \tau$ and set $q_A \leftarrow 0$ and $q_H \leftarrow 0$.

Figure 8.10: The wrapper that restricts access to the random oracle based on a given mining pattern.

we present this stronger ledger and show how such an implementation can be captured as a UC protocol which makes black-box use of the Ledger-Protocol to implement this ledger. The UC composition theorem makes such a proof immediate, as we do not need to think about the specifics of the invoked ledger protocol, and we can instead argue security in a world where this protocol is replaced by $\mathcal{G}_{\text{LEDGER}}^{\Bitcoin}$.

Protection of transactions using accounts. In Bitcoin, a miner creates an account ID AccountID by generating a signature key pair and hashing the public key. Any transaction of this party includes this account ID, i.e., $\mathtt{tx} = (\mathsf{AccountID}, \mathtt{tx}')$. An important property is that a transaction of a certain account cannot be invalidated by a transaction with a different account ID. Hence, to protect the validity of a transaction, upon submitting tx, party P_i has to sign it, append the signature and verification key to get a transaction $((\mathsf{AccountID}, \mathtt{tx}'), vk, \sigma)$. The validation predicate now additionally has to check that the account ID is the hash of the public key and that the signature σ is valid with respect to the verification key vk. Roughly, an adversary can invalidate tx, only by either forging a signature relative to vk, or by possessing key pair whose hash of the public key collides with the account ID of the honest party.

The realized ledger abstraction, denoted by $\mathcal{G}_{\text{LEDGER}}^{\Bitcoin+}$, is a ledger functionality as the one from the previous section, but which additionally allows parties to create unique accounts. Upon receiving a transaction from party P_i, $\mathcal{G}_{\text{LEDGER}}^{\Bitcoin+}$ only accepts a transaction containing the AccountID that was previously associated to P_i and ensures that parties are restricted to issue transactions using their own accounts. As we explain, this also amplifies transaction liveness.

8.8.1 A Stronger Ledger with Account Management

To achieve stronger guarantees than our original Bitcoin ledger, a party issues transactions relative to an account. More abstractly speaking, a transaction contains an identifier, AccountID, which can be seen as the abstract identity that claims ownership of the transaction. More specifically, we can represent this situation by having transactions tx be pairs $(\mathsf{AccountID}, \mathtt{tx}')$ with the above meaning. Signatures enter the picture at this level: an honest participant of the Bitcoin network will issue

only signed transactions on the network. In order to link verification key to the account, $\mathsf{AccountID}$ is the hash of the verification keys, where we require collision resistance. More concretely, whenever a miner is supposed to submit a transaction $\mathtt{tx}$, it signs it and appends the signature and its verification key. This bundle is distributed into the Bitcoin network. The validation consists now of three parts. First, it is verified that the public key matches the account, second, the signature is verified, and third, its validated whether the actual transaction $(\mathsf{AccountID}, \mathtt{tx}')$ is valid, with respect to a separate validation predicate $\mathsf{ValidTx}_{\mathrm{B}}$ on states and transactions $\mathtt{tx}$ of the above format. Only if all three tests succeed, the transactions is valid.

Looking ahead, the goal of this is the following: Assume that for the validation predicate $\mathsf{ValidTx}_{\mathrm{B}}$ it holds that if a transaction $(\mathsf{AccountID}, \mathtt{tx})$ is valid relative to a state, then the only reason why it can get invalid is due to the presence of another transaction with the same account. If we think of wallets, if a miner can spend his coins at current time, then only another transaction by himself can invalidate that (by spending the same coins, which the Bitcoin network will refuse). In combination with the unforgeability of signatures, no adversary can ever render a valid transaction invalid. Together with the weak liveness guarantee we can derive a better liveness guarantee.

We now show how to implement this account management in the $\mathcal{G}_{\mathrm{LEDGER}}^{\mathrm{B}}$ hybrid world to achieve a stronger ledger that formalizes account management in an ideal manner. Our protocol makes use of an existentially unforgeable digital signatures scheme.

The Protocol for Account Management

Hybrid ledger functionality. Let $\mathsf{ValidTx}_{\mathrm{B}}$ and $\mathsf{blockify}_{\mathrm{B}}$ be as in the previous section but with the following additional property: each transaction is a pair $\mathtt{tx} = (\mathsf{AccountID}, \mathtt{tx}')$ where the first part is bitstring of fixed length and the second part is an arbitrary transaction. In addition we require the following property: for any state $\mathtt{state}$ and any transaction $\mathtt{tx}$ it holds that $\mathsf{ValidTx}_{\mathrm{B}}(\mathtt{tx}, \mathtt{state}) = 1$ implies, for any state extension $\mathtt{state}||\mathtt{st}'$, that $\mathsf{ValidTx}_{\mathrm{B}}(\mathtt{tx}, \mathtt{state}||\mathtt{st}') = 1$, if $\mathtt{st}'$ does not contain a transaction with the same identifier $\mathsf{AccountID}$. Recall that we assume that Definition 8.5.1 is satisfied.

We assume the Bitcoin ledger functionality with the following vali-

dation predicate, which is defined relative to a collision-resistant hash function H, and a signature scheme DSS.

Algorithm to describe the assumed validation predicate

function $\mathsf{Val}_{\mathrm{DSS}}(\mathtt{BTX}, \mathtt{state}, \mathtt{buffer})$
  Let $\mathtt{BTX} = (\mathtt{tx}, \mathrm{txid}, \tau_L, p_i)$
  Parse $\mathtt{tx}$ as $((\mathsf{AccountID}, \mathtt{tx}'), vk, \sigma)$ (Return 0 in case of a wrong format)
  if $\mathsf{AccountID} = H(vk)$ **and** $\mathsf{Ver}(vk, \mathtt{tx}, \sigma) = 1$ **then**
    return $\mathsf{ValidTx}_{\mathcal{B}}(\mathtt{tx}, \mathtt{state})$
  else
    return 0

Protocol. The protocol is straightforward: whenever the protocol is given an input of the form $(\mathsf{AccountID}, \mathtt{tx})$ it first checks that it is the party associated with this account ID. Then, it receives the newest state from the ledger and checks, whether this input is valid with respect to the current state. If this is the case, the party signs the input and submits it to the ledger.

Protocol accountMgmt(P)

Initialization:

This protocol talks to the $\mathcal{G}_{\text{LEDGER}}$, but only changes the behavior of read or submit-queries to the ledger. Any other command is simply relayed to $\mathcal{G}_{\text{LEDGER}}$ and the corresponding output is given to the environment.
The protocol keeps a counter i and a vector $\mathsf{submitted}$ of inputs submitted to the ledger which are not yet contained in the state of the ledger.

Account Management:

- Upon receiving $(\text{CREATEACCOUNT}, \mathrm{sid})$, execute $(sk, vk) \leftarrow \mathsf{Gen}$, update $i \leftarrow i + 1$ and set $\mathsf{AccountID}_i \leftarrow H(vk)$. Return $(\text{CREATEACCOUNT}, \mathrm{sid}, \mathsf{AccountID}_i)$

Ledger Read and Write:

- Upon receiving $(\text{READ}, \mathrm{sid})$ send $(\text{READ}, \mathrm{sid})$ to $\mathcal{G}_{\text{LEDGER}}$ and receive as answer the current $\mathtt{state} = \mathtt{st}_1 || \ldots || \mathtt{st}_n$. Then do the following:
  $\mathtt{state}' \leftarrow \mathtt{st}_1$ ▷ Genesis state
  for $i = 2$ to n **do**
    From state block $\mathtt{st}_i$, extract the contents $(\mathtt{tx}_1, vk_1, \sigma_1) || \ldots || (\mathtt{tx}_n, vk_n, \sigma_n)$
    Define new block-content $\vec{x}' \leftarrow \mathtt{tx}_1 || \ldots || \mathtt{tx}_n$
    $\mathtt{state}' \leftarrow \mathtt{state} || \mathsf{blockify}_{\mathcal{B}}(\vec{x}')$
  Return $(\text{READ}, \mathrm{sid}, \mathtt{state}')$

- Upon receiving $(\text{SUBMIT}, \text{sid}, \texttt{tx})$, check that $\texttt{tx} = (\mathsf{AccountID}, \texttt{tx}')$ for $\mathsf{AccountID} \in \{\mathsf{AccountID}_1, \ldots \mathsf{AccountID}_i\}$. If the check fails, ignore the input. Otherwise, do the following:
  1. Read the state $\texttt{state}$ from $\mathcal{G}_{\text{LEDGER}}$ as above.
  2. If $\mathsf{ValidTx}_{\mathbb{B}}(\texttt{tx}, \texttt{state}) = 1$, then sign the input by $\sigma \leftarrow \mathsf{Sign}(sk, \texttt{tx})$ and send $(\text{SUBMIT}, \text{sid}, (\texttt{tx}, vk, \sigma))$

The Enhanced Ledger Functionality

We present an enhanced ledger functionality with a validation predicate that enforces that an adversarial transaction cannot prevent a transaction by an honest party to eventually make it into the stable state of the ledger. In particular, we get the following enhanced functionality:

Functionality $\mathcal{G}_{\text{LEDGER}}^{\ddot{\mathbb{B}}+}$

$\mathcal{G}_{\text{LEDGER}}^{\ddot{\mathbb{B}}+}$ is identical to $\mathcal{G}_{\text{LEDGER}}^{\ddot{\mathbb{B}}}$ except with the following additional capabilities:

Difference to standard Ledger:

- Upon receiving $(\text{CREATEACCOUNT}, \text{sid})$ from party P_i (or the adversary on behalf of a party P_i), send $(\text{ACCOUNTREQ}, \text{sid}, P_i)$ to $\mathcal{A}$ and upon receiving a reply $(\text{ACCOUNTREQ}, \text{sid}, P_i, \mathsf{AccountID})$ do the following:
  1. If $\mathsf{AccountID}$ is not yet associated to any party, store the pair $(\mathsf{AccountID}, P_i)$ internally and return $(\text{CREATEACCOUNT}, \text{sid}, \mathsf{AccountID})$ to P_i.
  2. If $\mathsf{AccountID}$ is already associated to a party, then output $(\text{CREATEACCOUNT}, \text{sid}, \text{Fail})$ to P_i.

Standard Bitcoin Ledger:

- Identical to $\mathcal{G}_{\text{LEDGER}}$ with validation predicate $\mathsf{Val}_{\text{strong}}$ and with the fixed transaction format described above. We omit the formal specification here.

The following validation predicate is used within $\mathcal{G}_{\text{LEDGER}}^{\ddot{\mathbb{B}}+}$ and provides better guarantees. We discuss the improvements in the next section.

Algorithm to define the strong validation

function $\mathsf{Val}_{\mathrm{strong}}(\mathtt{BTX}, \mathtt{state}, \mathtt{buffer})$
  Let $\mathtt{BTX} = (\mathtt{tx}, \mathrm{txid}, \tau_L, p_i)$
  if $\mathtt{tx} = (\mathsf{AccountID}, \mathtt{tx}')$ **and** AccountID is associated with p_i **then**
    return $\mathsf{ValidTx}_{\mathcal{B}}(\mathtt{tx}, \mathtt{state})$
  else
    return 0

We have the following lemma:

Lemma 8.8.1. *Let* DSS *be a secure digital signature scheme and let H be a collision resistant hash function. Then the protocol* accountMgmt *in the $\mathcal{G}_{\mathrm{LEDGER}}^{\mathcal{B}}$-hybrid world UC-realizes ledger $\mathcal{G}_{\mathrm{LEDGER}}^{\mathcal{B}+}$, where the functionalities are instantiated as described above.*

Proof Sketch. It is straightforward to write a simulator in the ideal-world execution that perfectly mimics the protocol as long as no hash-collision or signature forgery occurs. This is because the only non-trivial property that the ledger enforces (beyond what is already guaranteeed) is that just the account holder can submit a transaction but no one else in his name. If no hash-function collision is found, the only possible way is to forge a signature. If both events do not happen, the real world indeed implements the stronger validation predicate. Assuming a collision-resistant hash function a and signature scheme that is unforgeable under chosen-message attacks, this implies the statement. □

On the Better Guarantees

The stronger guarantee for honestly submitted transactions stems from two facts. First, by Definition 8.5.1, the state blocks contain transactions beyond coin-base transactions. Second, since a transaction of a party is associated with its account, and cannot be invalidated by another transaction with a different account, this implies that the transaction remains valid relative to `state` (unless the honest party itself issues a transaction that contradicts a previous transaction for one of its accounts, but we neglect this here). As an example, assume an honest party submits a single transaction for one of its accounts, and assume this transaction is valid relative to the state `state`. Then, by the defined enforcing mechanism of ExtendPolicy, this transaction is guaranteed to enter the

state after staying in the buffer for long enough, and when an honest party mines a subsequent block after this delay. This means that after that delay has passed, the transaction has to appear within the subsequent window of `windowSize` blocks.

A brief worst-case calculation. Looking at the ledger abstraction, we can directly compute the following worst-case upper bound for any miner (we neglect here the offset at the beginning of the execution for simplicity): after submitting the transaction, the transaction will appear (relative to the view of the submitting party) within the next $4 \cdot \texttt{windowSize}$ blocks after submitting the transaction (except with negligible probability). The reason is that upon submitting, (1) the view of the miner submitting the transaction could be `windowSize` blocks behind the head of the state of the ledger, (2) by the definition of ExtendPolicy, at most $2 \cdot \texttt{windowSize}$ blocks can be added to the state while the transaction is staying in the buffer before the ledger starts enforcing that the transaction be part of the subsequent next honest state block. This can be guaranteed within another interval of `windowSize` state blocks. We note that this calculation yields a quite loose bound. By the correspondence of `windowSize` and the chop-off parameter T of the Bitcoin protocol, and assuming that $T = 6$ blocks take approximately one hour, we get a worst-case time estimate for transaction liveness of four hours— given that transactions are correctly signed and are not invalidated due to other transactions with the same account.

Conclusion, Supplementary Material, and Bibliography

Chapter 9

Conclusion

This thesis is driven by one central question: what is security and how should it be defined? We saw that it is desirable to specify security statements as constructions, i.e., to formulate what is assumed and to give a precise description of the ideal system that a protocol should achieve based on the assumptions. The constructive cryptography framework is a perfect match to formulate and prove these kind of construction statements. We believe that our results help to understand and judge the importance of practically relevant protocols and to appreciate a systematic way to identify which ideal abstractions such protocols should emulate and to figure out in which cases existing game-based notions are (in-)sufficient for a given application.

We hope that the general approach of this thesis helps to spawn future research in various areas of cryptography. In light of this thesis, in the realm of secure communication, a general treatment of Internet protocols in constructive cryptography could help to build a more secure infrastructure. In the same way, the area of secure cloud storage could benefit from new analyses in a wide range of applications including for example the notions for proofs-of-ownership or de-duplication techniques. And finally, in the blockchain space, the emerging applications, some of which have the potential to become a critical part of the infrastructure in the information society, need careful analyses that smoothly support their secure development and design.

Appendix A

Details of Chapter 5

A.1 Finishing the Construction Proof

We conclude the last steps of the proof of Theorem 5.3.1 here, i.e., we complete the sequence of hybrid worlds as promised in Section 5.3.3. Recall that in the title of each box that depicts two hybrids at once, there are typically two names surrounded by solid or dashed boxes such as $\boxed{\mathbf{H}_0^{\mathcal{C}}}$ and $\boxed{\mathbf{H}_1^{\mathcal{C}}}$. This means that all code specifically surrounded by a dashed line is executed in $\mathbf{H}_0^{\mathcal{C}}$, but not $\mathbf{H}_1^{\mathcal{C}}$. Similarly, all code specifically surrounded by a solid line is executed in $\mathbf{H}_1^{\mathcal{C}}$, but not in $\mathbf{H}_0^{\mathcal{C}}$. All remaining code is executed in both systems. In cases where the box represents just one hybrid system, we might draw boxes to highlight certain parts of the code.

A.1.1 Completing Step 5.)

The fifth hybrid system $\mathbf{H}_4^{\mathcal{Z}}$ is a syntactic modification of the fourth. In particular, we observe that the mapping of identities to interfaces is stored redundantly, once within the network and once within the certification authority. Hence, it is sufficient to store it only once. We further simplify the case distinction upon input $(\texttt{send}, m, \mathsf{ID})$ at an interface $\mathtt{P}_i$, and upon input $(\texttt{inject}, s, \mathsf{ID}_s, \mathsf{ID}_r)$ at interface $\mathtt{E}$. The behavior of the resulting system is not affected by any of these changes and the two hybrids are equivalent, i.e., $\mathbf{H}_3^{\mathcal{Z}} = \mathbf{H}_4^{\mathcal{Z}}$.

Resource $\mathbf{H}_3^{\mathcal{Z}}$ and $\mathbf{H}_4^{\mathcal{Z}}$

Initialization

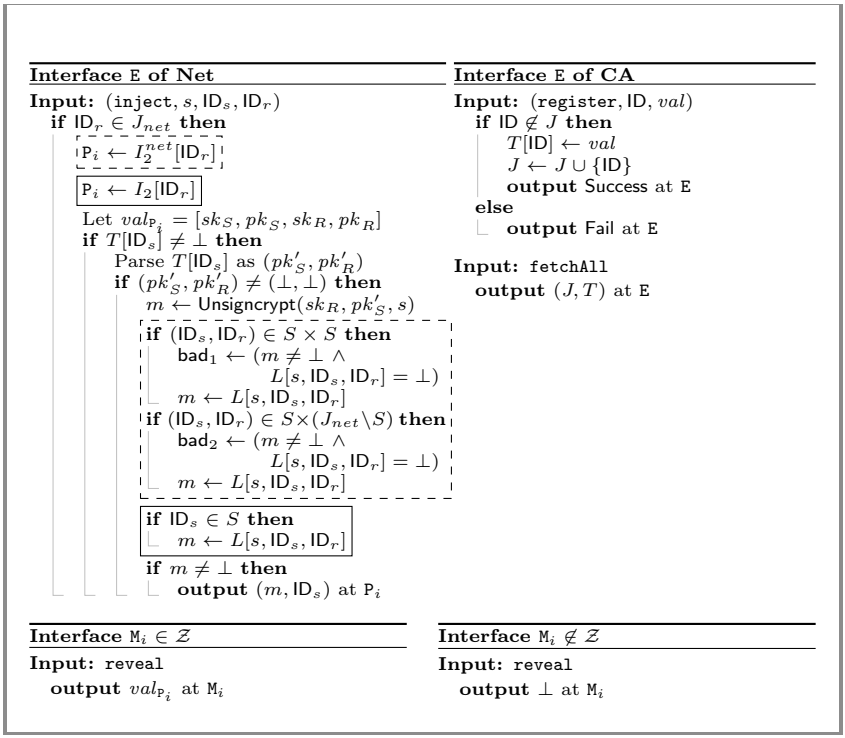
$J_{ca}, J_{net}, S \leftarrow \emptyset$
$I_1^{ca}, I_2^{ca}, I_1^{net}, I_2^{net}, T, L \leftarrow$ empty tables

$J_{ca}, J_{net}, S \leftarrow \emptyset$
$I_1, I_2, T, L \leftarrow$ empty tables

Interface $\mathtt{P}_i$

Input: $(\mathtt{register}, \mathsf{ID})$
  if $\mathsf{ID}_{\mathtt{P}_i} = \perp$ **then**
    $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S$
    $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R$
    if $\mathsf{ID} \notin J_{ca}$ **then**
      $J_{ca} \leftarrow J_{ca} \cup \{\mathsf{ID}\}$
      $I_1^{ca}[\mathtt{P}_i] \leftarrow \mathsf{ID}$
      $I_2^{ca}[\mathsf{ID}] \leftarrow \mathtt{P}_i$
      $T[\mathsf{ID}] \leftarrow (pk_S, pk_R)$
      if $\mathsf{ID} \notin J_{net} \wedge I_1[\mathtt{P}_i] = \perp$ **then**
        $J_{net} \leftarrow J_{net} \cup \{\mathsf{ID}\}$
        $I_1^{net}[\mathtt{P}_i] \leftarrow \mathsf{ID}$
        $I_2^{net}[\mathsf{ID}] \leftarrow \mathtt{P}_i$
        $I_1[\mathtt{P}_i] \leftarrow \mathsf{ID}$
        $I_2[\mathsf{ID}] \leftarrow \mathtt{P}_i$
        $\mathsf{ID}_{\mathtt{P}_i} \leftarrow \mathsf{ID}$
        $val_{\mathtt{P}_i} \leftarrow [sk_S, pk_S, sk_R, pk_R]$
        if $\mathtt{M}_i \notin \mathcal{Z}$ **then**
          $S \leftarrow S \cup \{\mathsf{ID}\}$
        output Success at $\mathtt{P}_i$
      else
        output Fail at $\mathtt{P}_i$
    else
      output Fail at $\mathtt{P}_i$

Input: $(\mathtt{send}, m, \mathsf{ID})$
  if $\mathsf{ID}_{\mathtt{P}_i} \neq \perp$ **then**
    Let $val_{\mathtt{P}_i} = [sk_S, pk_S, sk_R, pk_R]$
    if $T[\mathsf{ID}] \neq \perp$ **then**
      Parse $T[\mathsf{ID}]$ as (pk'_S, pk'_R)
      if $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**
        if $(\mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}) \in S \times S$ **then**
          $m^* \twoheadleftarrow \mathcal{M}$
          $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
        else if $(\mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}) \in (J_{net} \times S)$ **then**
          $m^* \twoheadleftarrow \mathcal{M}$
          $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
        else
          $m^* \leftarrow m$
          $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
        if $\mathsf{ID} \in S$ **then**
          $m^* \twoheadleftarrow \mathcal{M}$
          $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
        else
          $m^* \leftarrow m$
          $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
        $L[s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}] \leftarrow m$
        output $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ at $\mathtt{E}$



A.1.2 Completing Step 6.)

The sixth hybrid system $\mathbf{H}_5^{\mathcal{Z}}$ is a syntactic modification of the previous one. In this system, we observe that also the identities are stored redundantly, so it is sufficient to only store the identities in one set (which is J_{ca} in this case). Furthermore, we can test various conditions at once and do not need nested if-statements upon input $(\texttt{register}, \mathsf{ID})$ at an interface $\mathtt{P}_i$. The modifications we make in this step do not affect the behavior. Their sole purpose is to bring this system closer to the ideal world system. By inspecting the pseudo-code we conclude $\mathbf{H}_4^{\mathcal{Z}} = \mathbf{H}_5^{\mathcal{Z}}$.

Resource $\mathbf{H}_4^{\mathcal{Z}}$ and $\boxed{\mathbf{H}_5^{\mathcal{Z}}}$

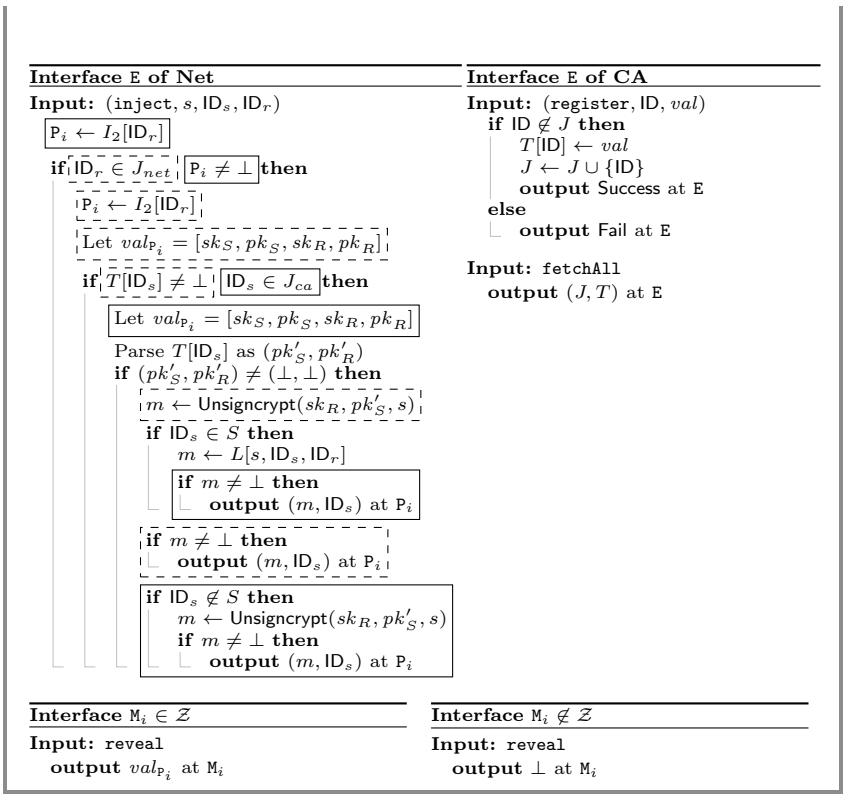
Initialization

$J_{ca}, J_{net}, S \leftarrow \emptyset$
$I_1, I_2, T, L \leftarrow$ empty tables

Interface $\mathtt{P}_i$

Input: $(\mathtt{register}, \mathsf{ID})$
if $\mathsf{ID}_{\mathtt{P}_i} = \bot$ $\boxed{\mathsf{ID} \notin J_{ca} \ \wedge \ I_1[\mathtt{P}_i] = \bot}$ **then**
  $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S$
  $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R$
  if $\mathsf{ID} \notin J_{ca}$ $\boxed{\textbf{or}\ \mathsf{true}}$ **then**
    $J_{ca} \leftarrow J_{ca} \cup \{\mathsf{ID}\}$
    $T[\mathsf{ID}] \leftarrow (pk_S, pk_R)$
    if $\mathsf{ID} \notin J_{net} \wedge I_1[\mathtt{P}_i] = \bot$ $\boxed{\textbf{or}\ \mathsf{true}}$ **then**
      $J_{net} \leftarrow J_{net} \cup \{\mathsf{ID}\}$
      $I_1[\mathtt{P}_i] \leftarrow \mathsf{ID}$
      $I_2[\mathsf{ID}] \leftarrow \mathtt{P}_i$
      $\mathsf{ID}_{\mathtt{P}_i} \leftarrow \mathsf{ID}$
      $val_{\mathtt{P}_i} \leftarrow [sk_S, pk_S, sk_R, pk_R]$
      if $\mathtt{M}_i \notin \mathcal{Z}$ **then**
        $S \leftarrow S \cup \{\mathsf{ID}\}$
      output Success at $\mathtt{P}_i$
    else
      output Fail at $\mathtt{P}_i$
  else
    output Fail at $\mathtt{P}_i$

Input: $(\mathtt{send}, m, \mathsf{ID})$
$\boxed{\mathsf{ID}_{\mathtt{P}_i} \leftarrow I_1[\mathtt{P}_i]}$
if $\mathsf{ID}_{\mathtt{P}_i} \neq \bot$ **then**
  Let $val_{\mathtt{P}_i} = [sk_S, pk_S, sk_R, pk_R]$
  if $T[\mathsf{ID}] \neq \bot$ $\boxed{\mathsf{ID} \in J_{ca}}$ **then**
    $\boxed{\text{Let } val_{\mathtt{P}_i} = [sk_S, pk_S, sk_R, pk_R]}$
    Parse $T[\mathsf{ID}]$ as (pk'_S, pk'_R)
    if $(pk'_S, pk'_R) \neq (\bot, \bot)$ **then**
      if $\mathsf{ID} \in S$ **then**
        $m^* \twoheadleftarrow \mathcal{M}$
        $\boxed{\begin{array}{l}\text{Let } \ell := |m| \\ m^* \twoheadleftarrow \{0,1\}^\ell\end{array}}$
        $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
      else
        $m^* \leftarrow m$
        $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
      $L[s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}] \leftarrow m$
      output $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ at $\mathtt{E}$



A.1.3 Completing Step 7.)

The seventh hybrid system $\mathbf{H}_6^{\mathcal{Z}}$ contains the final syntactic modifications and equals the ideal world with simulators attached at the corresponding interfaces. First, instead of generating the key pairs upon registration, we simply generate them when needed. For this, the system first generates some sufficiently long shared randomness. Whenever a key is generated for an honest party whose key is stolen, the corresponding part of the shared randomness is used to generate it (the simulator will learn the shared randomness for those people). If a key is to be generated for a party whose key is not stolen, the randomness is sampled uniformly at random to generate the keys. Finally, we replace the the list L that stores

messages, source and destination identities, and the ciphertext by two lists and implement an equivalent lookup using these two lists. Finally, we also rename the set J_{ca} to J and conclude that $\mathbf{H}_5^{\mathcal{Z}} = \mathbf{H}_6^{\mathcal{Z}}$.

Resource $\mathbf{H}_5^{\mathcal{Z}}$ and $\mathbf{H}_6^{\mathcal{Z}}$

Initialization

J_{ca} J J_{net}; $S \leftarrow \emptyset$ $j \leftarrow 0$

$I_1, I_2, T,$ L L_{sim}, L_{res} $\leftarrow$ empty tables

$(r_1^1, r_1^2)||(r_2^1, r_2^2)||\ldots||(r_n^1, r_n^2) \twoheadleftarrow \{0,1\}^{n\cdot(2\kappa)}$ ▷ Common randomness

Interface $\mathtt{P}_i$

Input: $(\mathtt{register}, \mathsf{ID})$

if $\mathsf{ID} \notin J_{ca}$ J $\wedge$ $I_1[\mathtt{P}_i] = \perp$ **then**

- $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S$
- $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R$
- J_{ca} $J \leftarrow J_{ca}$ J $\cup \{\mathsf{ID}\}$
- $T[\mathsf{ID}] \leftarrow (pk_S, pk_R)$
- $I_1[\mathtt{P}_i] \leftarrow \mathsf{ID}$
- $I_2[\mathsf{ID}] \leftarrow \mathtt{P}_i$
- $val_{\mathtt{P}_i} \leftarrow [sk_S, pk_S, sk_R, pk_R]$
- **if** $\mathtt{M}_i \notin \mathcal{Z}$ **then**
  - $S \leftarrow S \cup \{\mathsf{ID}\}$
- **output** Success at $\mathtt{P}_i$

else

- **output** Fail at $\mathtt{P}_i$

Input: $(\mathtt{send}, m, \mathsf{ID})$

$\mathsf{ID}_{\mathtt{P}_i} \leftarrow I_1[\mathtt{P}_i]$

if $\mathsf{ID}_{\mathtt{P}_i} \neq \perp$ **then**

if $\mathsf{ID} \in J_{ca}$ J **then**

for each $\mathsf{ID} \in J$ **do**

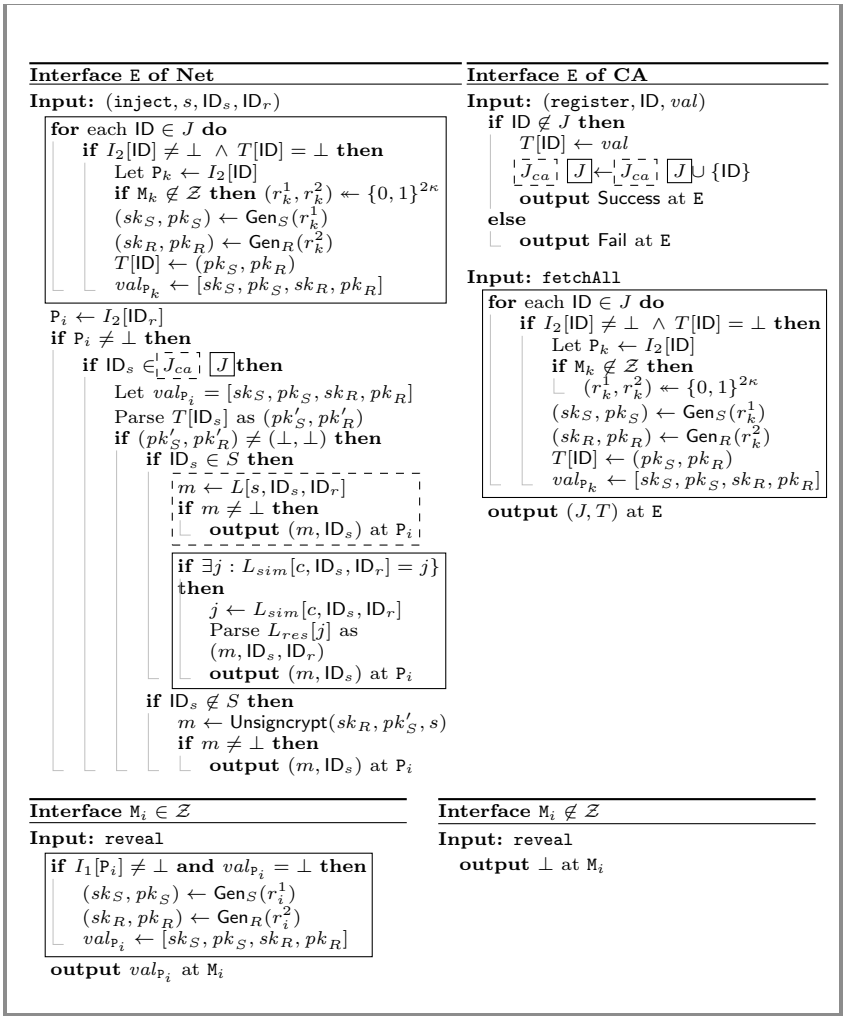
- **if** $I_2[\mathsf{ID}] \neq \perp \wedge T[\mathsf{ID}] = \perp$ **then**
  - Let $\mathtt{P}_k \leftarrow I_2[\mathsf{ID}]$
  - **if** $\mathtt{M}_k \notin \mathcal{Z}$ **then**
    - $(r_k^1, r_k^2) \twoheadleftarrow \{0,1\}^{2\kappa}$
    - $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S(r_k^1)$
    - $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R(r_k^2)$
    - $T[\mathsf{ID}] \leftarrow (pk_S, pk_R)$
    - $val_{\mathtt{P}_k} \leftarrow [sk_S, pk_S, sk_R, pk_R]$

Parse $val_{\mathtt{P}_i}$ as $[sk_S, pk_S, sk_R, pk_R]$

Parse $T[\mathsf{ID}]$ as (pk'_S, pk'_R)

if $(pk'_S, pk'_R) \neq (\perp, \perp)$ **then**

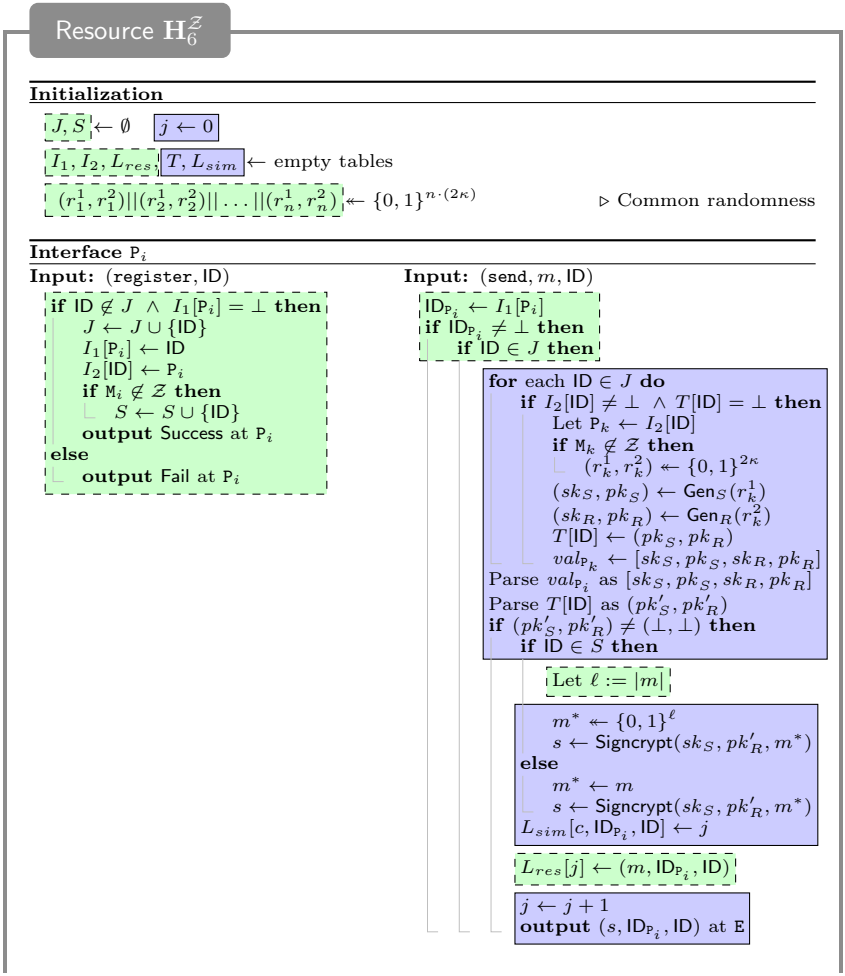
- **if** $\mathsf{ID} \in S$ **then**
  - Let $\ell := |m|$
  - $m^* \twoheadleftarrow \{0,1\}^\ell$
  - $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
- **else**
  - $m^* \leftarrow m$
  - $s \leftarrow \mathsf{Signcrypt}(sk_S, pk'_R, m^*)$
- $L[s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}] \leftarrow m$
- $L_{sim}[c, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID}] \leftarrow j$
- $L_{res}[j] \leftarrow (m, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$
- $j \leftarrow j + 1$
- **output** $(s, \mathsf{ID}_{\mathtt{P}_i}, \mathsf{ID})$ at $\mathtt{E}$



A.1.4 Completing Step 8.)

We depict the final hybrid system $\mathbf{H}_6^{\mathcal{Z}}$ a second time. For better accessibility, we color the corresponding parts of the simulators in blue and surround it by a solid line, and we color the code executed by the

constructed resource $\mathbf{SecNT}_n$ green and surround it by a dashed line. The last hybrid system is thus the compilation of the simulator and the constructed resource–and thus equals the ideal system—which can be concluded by inspection.



Interface E of Net

Input: $(\texttt{inject}, s, \mathsf{ID}_s, \mathsf{ID}_r)$

for each $\mathsf{ID} \in J$ **do**
  if $I_2[\mathsf{ID}] \neq \bot \ \wedge \ T[\mathsf{ID}] = \bot$ **then**
    Let $\mathtt{P}_k \leftarrow I_2[\mathsf{ID}]$
    if $\mathtt{M}_k \notin \mathcal{Z}$ **then** $(r_k^1, r_k^2) \twoheadleftarrow \{0,1\}^{2\kappa}$
    $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S(r_k^1)$
    $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R(r_k^2)$
    $T[\mathsf{ID}] \leftarrow (pk_S, pk_R)$
    $val_{\mathtt{P}_k} \leftarrow [sk_S, pk_S, sk_R, pk_R]$

$\mathtt{P}_i \leftarrow I_2[\mathsf{ID}_r]$
if $\mathtt{P}_i \neq \bot$ **then**
  if $\mathsf{ID}_s \in J$ **then**
    Let $val_{\mathtt{P}_i} = [sk_S, pk_S, sk_R, pk_R]$
    Parse $T[\mathsf{ID}_s]$ as (pk'_S, pk'_R)
    if $(pk'_S, pk'_R) \neq (\bot, \bot)$ **then**
      if $\mathsf{ID}_s \in S$ **then**
        if $\exists j : L_{sim}[c, \mathsf{ID}_s, \mathsf{ID}_r] = j\}$ **then**
          $j \leftarrow L_{sim}[c, \mathsf{ID}_s, \mathsf{ID}_r]$
        Parse $L_{res}[j]$ as $(m, \mathsf{ID}_s, \mathsf{ID}_r)$
        output (m, ID_s) at $\mathtt{P}_i$
      if $\mathsf{ID}_s \notin S$ **then**
        $m \leftarrow \mathsf{Unsigncrypt}(sk_R, pk'_S, s)$
        if $m \neq \bot$ **then**
          output (m, ID_s) at $\mathtt{P}_i$

Interface E of CA

Input: $(\texttt{register}, \mathsf{ID}, val)$

if $\mathsf{ID} \notin J$ **then**
  $T[\mathsf{ID}] \leftarrow val$
  $J \leftarrow J \cup \{\mathsf{ID}\}$
  output Success at E
else
  output Fail at E

Input: fetchAll

for each $\mathsf{ID} \in J$ **do**
  if $I_2[\mathsf{ID}] \neq \bot \ \wedge \ T[\mathsf{ID}] = \bot$ **then**
    Let $\mathtt{P}_k \leftarrow I_2[\mathsf{ID}]$
    if $\mathtt{M}_k \notin \mathcal{Z}$ **then**
      $(r_k^1, r_k^2) \twoheadleftarrow \{0,1\}^{2\kappa}$
    $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S(r_k^1)$
    $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R(r_k^2)$
    $T[\mathsf{ID}] \leftarrow (pk_S, pk_R)$
    $val_{\mathtt{P}_k} \leftarrow [sk_S, pk_S, sk_R, pk_R]$
output (J, T) at E

Interface $\mathtt{M}_i \in \mathcal{Z}$

Input: reveal

if $I_1[\mathtt{P}_i] \neq \bot$ **and** $val_{\mathtt{P}_i} = \bot$ **then**
  $(sk_S, pk_S) \leftarrow \mathsf{Gen}_S(r_i^1)$
  $(sk_R, pk_R) \leftarrow \mathsf{Gen}_R(r_i^2)$
  $val_{\mathtt{P}_i} \leftarrow [sk_S, pk_S, sk_R, pk_R]$
output $val_{\mathtt{P}_i}$ at $\mathtt{M}_i$

Interface $\mathtt{M}_i \notin \mathcal{Z}$

Input: reveal

output $\bot$ at $\mathtt{M}_i$

Appendix B

Details of Chapter 8

B.1 From Unicast to Multicast

A unicast channel can be defined as follows:

Functionality $\mathcal{F}_{\text{U-CH}}^{\Delta, P_R}$

The functionality is parametrized with a receiver P_R, and and upper bound Δ on the delay of any channel. It keeps track of the set of possible senders $\mathcal{P}$. Any newly registered (resp. deregistered) party is added to (resp. deleted from) $\mathcal{P}$. The list of messages is stored in $\vec{M}$, initially empty.

- Upon receiving (SEND, m) from some $P_s \in \mathcal{P}$ or from the adversary $\mathcal{A}$, choose a new unique message-ID midfor m, initialize variables $D_{\text{mid}} := 1$ and $D_{\text{mid}}^{MAX} = 1$, set $\vec{M} := \vec{M}||(m, \text{mid}, D_{\text{mid}})$, and send $(m, \text{mid}, D_{\text{mid}})$ to the adversary.
- Upon receiving (FETCH) from P_R:
  1. For all registered mids, set $D_{\text{mid}} := D_{\text{mid}} - 1$.
  2. Let $\vec{M}_0$ denote the subvector $\vec{M}$ including all triples $(m, \text{mid}, D_{\text{mid}})$ with $D_{\text{mid}} = 0$ (in the same order as they appear in $\vec{M}$). Delete all entries in $\vec{M}_0$ from $\vec{M}$ and send $\vec{M}_0$ to P_R.
- Upon receiving (DELAY, T_{mid}, mid) from the adversary, if $D_{\text{mid}}^{MAX} + T_{\text{mid}} \leq \Delta$ and mid is a message-ID registered in the current $\vec{M}$, set $D_{\text{mid}} := D_{\text{mid}} + T_{\text{mid}}$ and $D_{\text{mid}}^{MAX} := D_{\text{mid}}^{MAX} + T_{\text{mid}}$; otherwise, ignore the message.
- Upon receiving (SWAP, mid, mid$'$) from the adversary, if mid and mid$'$ are message-IDs registered in the current $\vec{M}$, then swap the triples $(m, \text{mid}, D_{\text{mid}})$ and $(m, \text{mid}', D_{\text{mid}'})$ in $\vec{M}$. Return (SWAP-OK) to the adversary.

B.1.1 On Realizing Multicast from Unicast

We sketch how to realize a multicast network, in particular its synchronized version along the lines of [KMTZ13], by means of a synchronized message-diffusion protocol over a network of unicast channels (and implicitly assuming a local clock to obtain the round structure). The core of this diffusion protocol are the assumed and known (e.g., by a common list of IP addresses) relay-nodes to which parties thus can connect and which forward in each round all new messages they received (either from registered parties or other relay nodes) in the previous round to all the unicast channels they are connected to as senders.[1] Let $G = (V, E)$ denote the (dynamically updatable) directed graph whose vertices V are the parties and the relay-nodes which are currently participating in the execution and an edge (p_i, p_j) is in E iff p_i is one of the senders of the multicast channel with receiver p_j. It is straightforward to verify that provided that G restricted to the honest parties (i.e., when corrupted parties and the edges that use them are deleted from G) remains *strongly connected* (i.e., there is a directed path between any two honest parties, in either direction), then the diffusion mechanism executed over unicast channels with delay at most Δ security realizes a multicast network with delay Δd where d is an upper bound of the diameter of G. Indeed, the simulator, which is given any message submitted to any unicast channel and enough activations when the dummy parties themselves get activated (note that it is essentially a synchronous computation among the relay-nodes), needs to simply simulate when the respective parties would see a message and schedule the corresponding deliveries by using the delays submitted by the adversary. The fact that each channel has at most Δ delay means that it will take delay at most ΔL rounds for it to travel through an honest path of length L. Last but not least, in order to receive messages from the network established this way, when a party joins the network, it has to multicast a special message to the relay-nodes that has to contain its identifier such that the relay-nodes can start sending messages to that party. This induces at most a delay of Δ rounds until the party is guaranteed to receive the messages sent over the network. For simplicity, we ignore this additional delay incurred by the registration to

[1] In order to ensure that parties can send some messages twice, a nonce is attached to each input message that is to be multicasted. The relayers do not add another nonce to the message they relay.

the network, and omit it in our specification of the multicast functionality in Section 8.2.2. If one implements the network using the above sketched method, one would formally obtain the a multicast functionality as given in Figure 8.1, but where the party set $\mathcal{P}$ contains all parties that have joined (and not yet left) the network at least Δ rounds ago, since the sketched solution does not support instant registration. All remaining guarantees remain unchanged with respect to this new party set.

B.2 Further Details on the Bitcoin Ledger

This section includes complementary material for Section Section 8.5. We here give the formal description of the Extend Policy for $\mathcal{G}^{\mathbb{B}}_{\text{LEDGER}}$ below. It is easy to observe that the computation performed by this algorithm is well-defined for any definition of Validate and Blockify.

The presentation is logically divided into the step of deriving a default extension and the actual tests whether the adversarial proposal is admissible. The default extension is taken as the ledger-state extension if and only the proposal by the adversary does not pass the test specified and implemented by ExtendPolicy. The derivation of the default extension is given as pseudo-code in Figure B.1. Note also that the policy makes the initial bootstrapping time of the chain explicit, where by bootstrapping time we mean the time it takes for the first state block to be inserted into the ledger state.

Algorithm ExtendPolicy for $\mathcal{G}^{\mathbb{B}}_{\text{LEDGER}}$

function ExtendPolicy($\vec{\mathcal{I}}^T_H$, `state`, `NxtBC`, `buffer`, $\vec{\tau}_{\texttt{state}}$)
We assume call-by-value and hence the function has no side effects.
This Function implements the Extend Policy of the Bitcoin Ledger.

$\vec{N}_{\texttt{df}} \leftarrow$ DefaultExtension($\vec{\mathcal{I}}^T_H$, `state`, `NxtBC`, `buffer`, $\vec{\tau}_{\texttt{state}}$) ▷ Extension if adversary violates policy.
Let τ_L be current ledger time (computed from $\vec{\mathcal{I}}^T_H$)
Parse `NxtBC` as a vector $((\text{hFlag}_1, \texttt{NxtBC}_1), \cdots, (\text{hFlag}_n, \texttt{NxtBC}_n))$
$\vec{N} \leftarrow \varepsilon$ ▷ Initialize Result
if $|\texttt{state}| \geq \texttt{windowSize}$ **then** ▷ Determine time of the block which is `windowSize` blocks behind the state head
  Set $\tau_{\texttt{low}} \leftarrow \vec{\tau}_{\texttt{state}}[|\texttt{state}| - \texttt{windowSize} + 1]$
else
  Set $\tau_{\texttt{low}} \leftarrow 0$

oldValidTxMissing $\leftarrow$ false ▷ Flag to keep track whether old enough, valid transactions are inserted.
for each list $\texttt{NxtBC}_i$ of transaction IDs **do** ▷ Compute the next state block and verify validity
  $\vec{N}_i \leftarrow \varepsilon$
  Use the txid contained in $\texttt{NxtBC}_i$ to determine the list of transactions
  Let $\vec{\texttt{tx}} = (\texttt{tx}_1, \ldots, \texttt{tx}_{|\texttt{NxtBC}_i|})$ denote the transactions of $\texttt{NxtBC}_i$
  if $\texttt{tx}_1$ is not a coin-base transaction **then**
    return $\vec{N}_{\texttt{df}}$
  else
    $\vec{N}_i \leftarrow \texttt{tx}_1$
    for $j = 2$ to $|\texttt{NxtBC}_i|$ **do**
      Set $\texttt{st}_i \leftarrow \mathsf{blockify}_{\mathcal{B}}(\vec{N}_i)$
      if $\mathsf{ValidTx}_{\mathcal{B}}(\texttt{tx}_j, \texttt{state}||\texttt{st}_i) = 0$ **then**
        return $\vec{N}_{\texttt{df}}$ ▷ Default Extension if adversarial proposal is invalid
      $\vec{N}_i \leftarrow \vec{N}_i||\texttt{tx}_j$
    Set $\texttt{st}_i \leftarrow \mathsf{blockify}_{\mathcal{B}}(\vec{N}_i)$
  if the proposal is declared to be an honest block, i.e., $\text{hFlag}_i = 1$ **then**
    for each $\texttt{BTX} = (\texttt{tx}, \text{txid}, \tau', P_i) \in \texttt{buffer}$ of an honest party P_i with time $\tau' < \tau_{\texttt{low}} - \frac{\texttt{Delay}}{2}$ **do**
      if $\mathsf{ValidTx}_{\mathcal{B}}(\texttt{tx}, \texttt{state}||\texttt{st}_i) = 1$ but $\texttt{tx} \notin \vec{N}_i$ **then**
        oldValidTxMissing $\leftarrow$ true ▷ A transaction is missing in adversarial proposal.
  $\vec{N} \leftarrow \vec{N}||\vec{N}_i$
  $\texttt{state} \leftarrow \texttt{state}||\texttt{st}_i$
  $\vec{\tau}_{\texttt{state}} \leftarrow \vec{\tau}_{\texttt{state}}||\tau_L$
  $j \leftarrow \max\{\{\texttt{windowSize}\} \cup \{k \mid \texttt{st}_k \in \texttt{state} \wedge \text{proposal of } \texttt{st}_k \text{ had hFlag} = 1\}\}$
  ▷ Determine most recent honestly-generated block in the interval behind the head.
  if $|\texttt{state}| - j \geq \eta$ **then**
    return $\vec{N}_{\texttt{df}}$ ▷ Adversary proposed too few honestly generated blocks.
  if $|\texttt{state}| \geq \texttt{windowSize}$ **then**
    ▷ Update $\tau_{\texttt{low}}$: the time of the state block which is $\texttt{windowSize}$ blocks behind the head of the current, possibly extended state
    Set $\tau_{\texttt{low}} \leftarrow \vec{\tau}_{\texttt{state}}[|\texttt{state}| - \texttt{windowSize} + 1]$
  else
    Set $\tau_{\texttt{low}} \leftarrow 0$
if $\tau_L - \tau_{\texttt{low}} < \texttt{minTime}_{\texttt{window}}$ **then** ▷ Ensure that ledger does not proceed too fast
  return ε
else if $\tau_{\texttt{low}} > 0$ **and** $\tau_L - \tau_{\texttt{low}} > \texttt{maxTime}_{\texttt{window}}$ **then** ▷ A sequence of blocks cannot take too much time.
  return $\vec{N}_{\texttt{df}}$
else if $\tau_{\texttt{low}} = 0$ **and** $\tau_L - \tau_{\texttt{low}} > 2 \cdot \texttt{maxTime}_{\texttt{window}}$ **then** ▷ Bootstrapping cannot take too much time.
  return $\vec{N}_{\texttt{df}}$
else if oldValidTxMissing **then** ▷ If not all old enough, valid transactions have been included.
  return $\vec{N}_{\texttt{df}}$
return $\vec{N}$

Algorithm for Default State Extension

function DefaultExtension($\vec{\mathcal{I}}_H^T$, state, NxtBC, buffer, $\vec{\tau}_{\texttt{state}}$)
We assume call-by-value and hence the function has no side effects.
The function returns a policy-compliant extension of the ledger state.

Let τ_L be current ledger time (computed from $\vec{\mathcal{I}}_H^T$)
Set $\vec{N}_{\texttt{df}} \leftarrow \texttt{tx}^{\text{coin-base}}_{\text{minerID}}$ of an honest miner
Sort buffer according to time stamps and let $\vec{\texttt{tx}} = (\texttt{tx}_1, \ldots, \texttt{tx}_n)$ be the transactions in buffer
Set st $\leftarrow \mathsf{blockify}_{\mathbb{B}}(\vec{N}_{\texttt{df}})$
repeat
  Let $\vec{\texttt{tx}} = (\texttt{tx}_1, \ldots, \texttt{tx}_n)$ be the current list of (remaining) transactions
  for $i = 1$ to n **do**
    if $\mathsf{ValidTx}_{\mathbb{B}}(\texttt{tx}_i, \texttt{state}||\texttt{st}) = 1$ **then**
      $\vec{N}_{\texttt{df}} \leftarrow \vec{N}_{\texttt{df}}||\texttt{tx}_i$
      Remove $\texttt{tx}_i$ from $\vec{\texttt{tx}}$
      Set st $\leftarrow \mathsf{blockify}_{\mathbb{B}}(\vec{N}_{\texttt{df}})$
until $\vec{N}_{\texttt{df}}$ does not increase anymore
if $|\texttt{state}| + 1 \geq \texttt{windowSize}$ **then** ▷ Let $\tau_{\texttt{low}}$ be the time of the block which is windowSize $-$ 1 blocks behind the head of the state.
  Set $\tau_{\texttt{low}} \leftarrow \vec{\tau}_{\texttt{state}}[|\texttt{state}| - \texttt{windowSize} + 2]$
else
  Set $\tau_{\texttt{low}} \leftarrow 0$
$c \leftarrow 1$
while $\tau_L - \tau_{\texttt{low}} > \texttt{maxTime}_{\texttt{window}}$ **do**
  Set $\vec{N}_c \leftarrow \texttt{tx}^{\text{coin-base}}_{\text{minerID}}$ of an honest miner
  $\vec{N}_{\texttt{df}} \leftarrow \vec{N}_{\texttt{df}}||\vec{N}_c$
  $c \leftarrow c + 1$
  if $|\texttt{state}| + c \geq \texttt{windowSize}$ **then** ▷ Update $\tau_{\texttt{low}}$ to the time of the state block which is windowSize $- c$ blocks behind the head.
    Set $\tau_{\texttt{low}} \leftarrow \vec{\tau}_{\texttt{state}}[|\texttt{state}| - \texttt{windowSize} + c + 1]$
  else
    Set $\tau_{\texttt{low}} \leftarrow 0$
return $\vec{N}_{\texttt{df}}$

Figure B.1: Function to compute a policiy-compliant default ledger-state extension.

B.3 Further Details on Modularization of the Ledger Protocol

B.3.1 The Modular Ledger Protocol

We describe a modularized version of the UC Bitcoin protocol which is indistinguishable from the original protocol:

Protocol Modular-Ledger-Protocol$_T(P)$

Variables and Initial Values:

- The same as in the original protocol, except replace:

  The protocol stores a local (working) chain $\mathcal{C}_{loc}$ which initially contains the genesis block, i.e., $\mathcal{C}_{loc} \leftarrow (\mathbf{G})$.

  by

  The protocol manages the exported ledger state $\vec{st}_{\text{exp}}$ which initially is the genesis state, i.e. , $\vec{st} \leftarrow (\mathit{gen})$. It also manages a local (working) state $\vec{st}_{\text{loc}}$ (initially also the genesis state).

Registration/De-Registration:

- As in the original protocol, but where the two local setup functionalities $(\mathcal{F}_{\text{N-MC}}^{\text{bc}}, \mathcal{F}_{\text{RO}})$ are subsumed by one local functionality $\mathcal{F}_{\text{StX}}$.

Ledger-Queries:
Ledger queries are only answered once registered.

- As in the original protocol.

Handling other external calls:

- As in the original protocol.

Furthermore, the only places where we modify the original protocol in a non-trivial way are in the main sub-processes (only executed once registered):

Sub-Protocol ExtendState(st)

Send $(\text{SUBMIT-NEW}, \text{sid}, \vec{\mathtt{st}}_{\text{loc}}, \mathtt{st})$ to $\mathcal{F}_{\text{StX}}$.
Denote the response by $(\text{SUCCESS}, \text{sid}, B)$ of $\mathcal{F}_{\text{StX}}$.
if $B = 1$ **then**
  Update the local state, i.e., $\vec{\mathtt{st}}_{\text{loc}} \leftarrow \vec{\mathtt{st}}_{\text{loc}} || \mathtt{st}$.
Send $(\text{CONTINUE}, \text{sid})$ to $\mathcal{F}_{\text{StX}}$ ▷ Broadcast current state using $\mathcal{F}_{\text{StX}}$.

and

Sub-Protocol FetchInformation

Send $(\text{FETCH-NEW}, \text{sid})$ to $\mathcal{F}_{\text{StX}}$.
Denote the response from $\mathcal{F}_{\text{StX}}$ by $(\text{FETCH-NEW}, \text{sid}, (\vec{\mathtt{st}}_1, \ldots, \vec{\mathtt{st}}_k))$.
Set both $\vec{\mathtt{st}}_{\text{loc}}$,$\vec{\mathtt{st}}_{\text{exp}}$ to the longest state in $\vec{\mathtt{st}}_{\text{loc}}, \vec{\mathtt{st}}_{\text{exp}}, \vec{\mathtt{st}}_1, \ldots, \vec{\mathtt{st}}_k$ (to resolve ties the ordering decides).
Send $(\text{FETCH}, \text{sid})$ to $\mathcal{F}^{\mathtt{tx}}_{\text{N-MC}}$; denote the response from $\mathcal{F}^{\mathtt{tx}}_{\text{N-MC}}$ by $(\text{FETCH}, \text{sid}, b)$.
Extract received transactions $(\mathtt{tx}_1, \ldots, \mathtt{tx}_k)$ from b.
Set $\mathtt{buffer} \leftarrow \mathtt{buffer} || (\mathtt{tx}_1, \ldots, \mathtt{tx}_k)$.
If a NEW-PARTY message was received, set $\text{WELCOME} \leftarrow 1$. Otherwise, set $\text{WELCOME} \leftarrow 0$.
Remove all transactions from $\mathtt{buffer}$ which are invalid with respect to $\vec{\mathtt{st}}^{\lceil T}_{\text{loc}}$

B.3.2 On the Soundness of the Modular Decomposition

We perform a "game-hopping" argument to show that Ledger-Protocol UC emulates the protocol Modular-Ledger-Protocol when in the latter protocol, the invocations to $\mathcal{F}_{\text{StX}}$ are replace by calls to sub-process StateExchange-Protocol. We start with the original Ledger-Protocol and consider the protocol part below where will alter the protocol step by step.

Fragments of Original Protocol Part

Initialization:
The protocol stores a local (working) chain $\mathcal{C}_{loc}$ which initially contains the genesis block, i.e., $\mathcal{C}_{loc} \leftarrow (\mathbf{G})$. [...]

ExtendState(st):
$\mathcal{C}_{\text{new}} \leftarrow \mathsf{extendchain}_{\mathtt{D}}(\mathcal{C}_{loc}, \mathtt{st}, q)$
if $\mathcal{C}_{\text{new}} \neq \mathcal{C}_{loc}$ **then**

Update the local chain, i.e., $\mathcal{C}_{loc} \leftarrow \mathcal{C}_{\text{new}}$.
Send $(\text{MULTICAST}, \text{sid}, \mathcal{C}_{loc})$ to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ ▷ Multicast current chain

FetchInformation:

▷ Update the local state
Send $(\text{FETCH}, \text{sid})$ to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$; denote the response from $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ by $(\text{FETCH}, \text{sid}, b)$.
Extract valid chains $\mathcal{C}_1, \ldots, \mathcal{C}_k$ from b.
Set both $\mathcal{C}_{loc}, \mathcal{C}_{\text{exp}}$ to the longest valid chain in $\mathcal{C}_{loc}, \mathcal{C}_{\text{exp}}, \mathcal{C}_1, \ldots, \mathcal{C}_k$ (to resolve ties the ordering decides).
[...]

Modification 1. The first modification of the protocol (see below) proceeds as Ledger-Protocol except (a) it stores a history of all valid chains in a tree $\mathcal{T}$ and (b) in the **ExtendState(st)** procedure it checks that $\vec{\mathtt{st}}||\mathtt{st}$ is a valid state and that there exists a chain in $\mathcal{T}$ which encodes the state $\vec{\mathtt{st}}$. We observe that the protocol calls **ExtendState(st)** only with $\mathtt{st}$ where $\vec{\mathtt{st}}||\mathtt{st}$ is a valid state. This implies that the first check is always satisfied. Moreover, note that the current local chain $\mathcal{C}_{loc}$ which encodes state $\vec{\mathtt{st}}$ is at any time stored in the tree $\mathcal{T}$. We therefore call the state encoded in $\mathcal{C}_{loc}$ by $\vec{\mathtt{st}}_{\text{loc}}$ and see that the second check is therefore also always satisfied. Hence, the modified protocol has the same input/output behavior as the Ledger-Protocol.

Fragments, Modification 1

Initialization:

The protocol stores a local (working) chain $\mathcal{C}_{loc}$ which initially contains the genesis block, i.e., $\mathcal{C}_{loc} \leftarrow (\mathbf{G})$. [...]
The protocol additionally maintains a tree $\mathcal{T}$ of valid chains which initially contains the (genesis) chain $(\mathbf{G})$.

ExtendState(st):

if $\mathsf{isvalidstate}_{\mathcal{B}}(\vec{\mathtt{st}}_{\text{loc}}||\mathtt{st}) = 1$ **then**
  if there exists $\mathcal{C} \in \mathcal{T}$ which encodes $\vec{\mathtt{st}}_{\text{loc}}$ **then**
    $\mathcal{C}_{\text{new}} \leftarrow \mathsf{extendchain}_{\mathcal{D}}(\mathcal{C}_{loc}, \mathtt{st}, q)$
    if $\mathcal{C}_{\text{new}} \neq \mathcal{C}_{loc}$ **then**
      Update the local chain, i.e., $\mathcal{C}_{loc} \leftarrow \mathcal{C}_{\text{new}}$.
      Add $\mathcal{C}_{loc}$ to $\mathcal{T}$
    Send $(\text{MULTICAST}, \text{sid}, \mathcal{C}_{\text{loc}})$ to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ ▷ Multicast current chain

FetchInformation:

Send $(\text{FETCH}, \text{sid})$ to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$; denote the response from $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ by $(\text{FETCH}, \text{sid}, b)$.
Extract all valid chains $\mathcal{C}_1, \ldots, \mathcal{C}_k$ from b and add them to $\mathcal{T}$.

Set both $\mathcal{C}_{loc}, \mathcal{C}_{\text{exp}}$ to the longest valid chain in $\mathcal{C}_{loc}, \mathcal{C}_{\text{exp}}, \mathcal{C}_1, \ldots, \mathcal{C}_k$ (to resolve ties the ordering decides).
[...]

Modification 2. In Modification 2 (see below) the local state $\vec{\mathtt{st}}_{\text{loc}}$ is stored directly instead of being encoded in chain $\mathcal{C}_{loc}$. The procedures **ExtendState(st)** and **FetchInformation** are modified to accommodate this change. Note that the $\mathcal{C}_{loc}$ is stored in $\mathcal{T}$ as we have seen in the first modification. This implies that the behavior of **ExtendState(st)** remains the same as in the first modification.

Fragments, Modification 2

Initialization:

The protocol manages [...] a local (working) state $\vec{\mathtt{st}}_{\text{loc}}$ (initially also the genesis state).[...]
The protocol additionally maintains a tree $\mathcal{T}$ of valid chains which initially contains the genesis chain (**G**).

ExtendState(st):

if $\mathsf{isvalidstate}_{\mathcal{G}}(\vec{\mathtt{st}}_{\text{loc}}||\mathtt{st}) = 1$ **then**
  if there exists $\mathcal{C} \in \mathcal{T}$ which encodes $\vec{\mathtt{st}}_{\text{loc}}$ **then**
    $\mathcal{C}_{\text{new}} \leftarrow \mathsf{extendchain}_{\mathtt{D}}(\mathcal{C}, \mathtt{st}, q)$
    if $\mathcal{C}_{\text{new}} \neq \mathcal{C}$ **then**
      Add $\mathcal{C}$ to $\mathcal{T}$
      Update the local state, i.e., $\vec{\mathtt{st}}_{\text{loc}} \leftarrow \vec{\mathtt{st}}_{\text{loc}}||\mathtt{st}$.
    Send $(\textsc{multicast}, \text{sid}, \mathcal{C}_{\text{loc}})$ to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ ▷ Multicast current chain

FetchInformation:

Send $(\textsc{fetch}, \text{sid})$ to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$; denote the response from $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ by $(\textsc{fetch}, \text{sid}, b)$.
Extract all valid chains $\mathcal{C}_1, \ldots, \mathcal{C}_k$ from b and add them to $\mathcal{T}$.
Extract all state $\vec{\mathtt{st}}_1, \ldots, \vec{\mathtt{st}}_k$ from chains $\mathcal{C}_1, \ldots, \mathcal{C}_k$.
Set both $\vec{\mathtt{st}}_{\text{loc}}, \vec{\mathtt{st}}_{\text{exp}}$ to the longest state in $\vec{\mathtt{st}}_{\text{loc}}, \vec{\mathtt{st}}_{\text{exp}}, \vec{\mathtt{st}}_1, \ldots, \vec{\mathtt{st}}_k$ (to resolve ties the ordering decides).
[...]

Modification 3. In Modification 3 (see below) parts of the procedures **ExtendState(st)** and **FetchInformation** are split off into separate sub-procedures. Otherwise the protocol remains the same. As there are no changes to the program logic the protocol still has the same behavior as the original protocol.

Fragments, Modification 3

Initialization:

The protocol manages [...] a local (working) state $\vec{\mathtt{st}}_{\mathrm{loc}}$ (initially also the genesis state).[...]
The protocol additionally maintains a tree $\mathcal{T}$ of valid chains which initially contains the (genesis) chain $(\mathbf{G})$.

ExtendState(st):

$B \leftarrow$ SUBMIT-NEW$(\vec{\mathtt{st}}_{\mathrm{loc}}, \mathtt{st})$
if $B = 1$ **then**
  Update the local state, i.e., $\vec{\mathtt{st}}_{\mathrm{loc}} \leftarrow \vec{\mathtt{st}}_{\mathrm{loc}} || \mathtt{st}$.
Execute CONTINUE. ▷ Broadcast current chain

Procedure SUBMIT-NEW$(\vec{\mathtt{st}}, \mathtt{st})$:
if $\mathsf{isvalidstate}_{\hat{\mathcal{B}}}(\vec{\mathtt{st}} || \mathtt{st}) = 1$ **then**
  if there exists $\mathcal{C}' \in \mathcal{T}$ which encodes $\vec{\mathtt{st}}$ **then**
    Set $\mathcal{C} \leftarrow \mathcal{C}'$. ▷ $\mathcal{C}$ is assumed to be a global variable
    $\mathcal{C}_{\mathrm{new}} \leftarrow \mathsf{extendchain}_{\mathtt{D}}(\mathcal{C}, \mathtt{st}, q)$
    if $\mathcal{C}_{\mathrm{new}} \neq \mathcal{C}$ **then**
      Add $\mathcal{C}$ to $\mathcal{T}$
      return 1
    return 0

Procedure CONTINUE:
Send $(\textsc{multicast}, \mathrm{sid}, \mathcal{C})$ to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$

FetchInformation:

$(\vec{\mathtt{st}}_1, \ldots, \vec{\mathtt{st}}_k) \leftarrow$ FETCH-NEW
Set both $\vec{\mathtt{st}}_{\mathrm{loc}}, \vec{\mathtt{st}}_{\mathrm{exp}}$ to the longest state in $\vec{\mathtt{st}}_{\mathrm{loc}}, \vec{\mathtt{st}}_{\mathrm{exp}}, \vec{\mathtt{st}}_1, \ldots, \vec{\mathtt{st}}_k$ (to resolve ties the ordering decides).
[...]

Procedure FETCH-NEW:
Send $(\textsc{fetch}, \mathrm{sid})$ to $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$; denote the response from $\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}$ by $(\textsc{fetch}, \mathrm{sid}, b)$.
Extract all valid chains $\mathcal{C}_1, \ldots, \mathcal{C}_k$ from b and add them to $\mathcal{T}$.
Extract states $\vec{\mathtt{st}}_1, \ldots, \vec{\mathtt{st}}_s$ from $\mathcal{C}_1, \ldots, \mathcal{C}_k$ and output them.

Final Considerations. Finally consider the part of Modular-Ledger-Protocol below which is the same as Modification 3 except that the chain storage $\mathcal{T}$ and the calls to sub-procedures SUBMIT-NEW, CONTINUE, and FETCH-NEW are replaced by the calls to $\mathcal{F}_{\mathrm{STX}}$. Now, if these calls are answered by the protocol StateExchange-Protocol, we get the exact same behavior as implemented by the third modification above. To see this, we recap quickly the relevant fragment:

Fragments, Modular-Ledger-Protocol

Initialization:

The protocol manages [...] a local (working) state $\vec{\mathtt{st}}_{\mathrm{loc}}$ (initially also the genesis state). [...]

ExtendState(st):

Send $(\textsc{submit-new}, \mathrm{sid}, \vec{\mathtt{st}}_{\mathrm{loc}}, \mathtt{st})$ to $\mathcal{F}_{\mathrm{StX}}$.
Denote the response by $(\textsc{success}, \mathrm{sid}, B)$ of $\mathcal{F}_{\mathrm{StX}}$.
if $B = 1$ **then**
  Update the local state, i.e., $\vec{\mathtt{st}}_{\mathrm{loc}} \leftarrow \vec{\mathtt{st}}_{\mathrm{loc}} || \mathtt{st}$.
Send $(\textsc{continue}, \mathrm{sid})$ to $\mathcal{F}_{\mathrm{StX}}$ ▷ Broadcast current state using $\mathcal{F}_{\mathrm{StX}}$.

FetchInformation:

Send $(\textsc{fetch-new}, \mathrm{sid})$ to $\mathcal{F}_{\mathrm{StX}}$.
Denote the response from $\mathcal{F}_{\mathrm{StX}}$ by $(\textsc{fetch-new}, \mathrm{sid}, (\vec{\mathtt{st}}_1, \ldots, \vec{\mathtt{st}}_k))$.
Set $\vec{\mathtt{st}}_{\mathrm{loc}}, \vec{\mathtt{st}}_{\mathrm{exp}}$ to the longest state in $\vec{\mathtt{st}}_{\mathrm{loc}}, \vec{\mathtt{st}}_{\mathrm{exp}}, \vec{\mathtt{st}}_1, \ldots, \vec{\mathtt{st}}_k$ (to resolve ties the ordering decides).
[...]

Also, when registering/de-registering from $\mathcal{F}_{\mathrm{StX}}$, StateExchange-Protocol simply registers/de-registers from $(\mathcal{F}_{\text{N-MC}}^{\mathsf{bc}}, \mathcal{F}_{\mathrm{RO}})$, just as the original protocol. As we prove in the main body in Lemma 8.6.1, StateExchange-Protocol UC-realizes $\mathcal{F}_{\mathrm{StX}}$, hence replacing calls to StateExchange-Protocol by calls to the ideal (hybrid) functionality yields an indistinguishable protocol to Ledger-Protocol.

B.4 The Simulator of the Main Theorem

It follows the formal specification of the simulator.

Simulator $\mathcal{S}_{\text{ledg}}$

Initialization:

The simulator manages internally a simulated state-exchange functionality $\mathcal{F}_{\text{StX}}$, a simulated network $\mathcal{F}_{\text{N-MC}}$. An honest miner P registered to $\mathcal{G}^{\mathbb{B}}_{\text{LEDGER}}$ is assumed to be registered in all simulated functionalities. Moreover, the simulator maintains the local state $\vec{\mathtt{st}}_P$ and the buffer of transactions $\mathtt{buffer}_P$ of such a party. Upon any activation, the simulator will query the current party set from the ledger (and simulate the corresponding message they send out to the network in the first maintain-ledger activation after registration), query all activations from honest parties $\vec{\mathcal{I}}^T_H$, and read the current clock value to learn the time. In particular, the simulator knows which parties are honest and synchronized and which parties are de-synchronized.

General Structure:
The simulator internally runs adversary $\mathcal{A}$ in a black-box way and simulates the interaction between $\mathcal{A}$ and the (emulated) real-world hybrid functionalities. The inputs from $\mathcal{A}$ to the clock are simply relayed (and the replies given back to $\mathcal{A}$). The ideal world consists of the ledger functionality and the clock.

Messages from the Clock:

- Upon receiving $(\text{CLOCK-UPDATE}, \text{sid}_C, P)$ from $\mathcal{G}_{\text{CLOCK}}$, if P is an honest registered party, then remember that this party has received such a clock update (and the environment gets an activation). Otherwise, send $(\text{CLOCK-UPDATE}, \text{sid}_C, P)$ to $\mathcal{A}$. In addition (before releasing the activation token), the simulator checks whether the clock advances. If so, and if this was a working mini-round (and hence all maintain commands have already been submitted by honest and synchronized parties), then execute ExtendLedgerState before giving the activation to $\mathcal{A}$.

Messages from the Ledger:

- Upon receiving $(\text{SUBMIT}, \mathtt{BTX})$ from $\mathcal{G}^{\mathbb{B}}_{\text{LEDGER}}$ where $\mathtt{BTX} := (\mathtt{tx}, \text{txid}, \tau, P)$ forward $(\text{MULTICAST}, \text{sid}, \mathtt{tx})$ to the simulated network $\mathcal{F}_{\text{N-MC}}$ in the name of P. Output the answer of $\mathcal{F}_{\text{N-MC}}$ to the adversary.

- Upon receiving $(\text{MAINTAIN-LEDGER}, \text{sid}, \text{minerID})$ from $\mathcal{G}^{\mathbb{B}}_{\text{LEDGER}}$, extract from $\vec{\mathcal{I}}^T_H$ the party P_i that issued this query. If P_i has already done its instructions for the current mini-round, then ignore the request. Otherwise, do:

  1. Execute $\text{SimulateMining}(P_{\text{minerID}}, \tau)$ and if this was the last maintain command in a working mini-round and the round will advance, then execute ExtendLedgerState before giving the activation to $\mathcal{A}$.
  2. In addition, remember that party P_i is done with mining in the current mini-round.

- Upon any further activation of the simulator, the simulator inspects the entire sequence of inputs by honest parties to the ledger $\vec{\mathcal{I}}^T_H$ and does the following:

1. For any input, $I = (\text{READ}, \text{sid})$ of party P, if the current round is an update mini-round, then execute Step 4 of the mining procedure as below in SIMULATEMINING
2. Remember that the update for party P is done for this round.

Simulation of the State Exchange Functionality:

- Upon receiving $(\text{SUBMIT-NEW}, \text{sid}, \vec{\mathtt{st}}, \mathtt{st})$ from $\mathcal{A}$ on behalf of a corrupted $P \in \mathcal{P}_{stx}$, then relay it to the simulated $\mathcal{F}_{\text{STX}}$ and do the following:
  1. If $\mathcal{F}_{\text{STX}}$ returns $(\text{SUCCESS}, B)$ give this reply to $\mathcal{A}$
  2. If $\mathcal{A}$ replies with $(\text{CONTINUE}, \text{sid})$, input $(\text{CONTINUE}, \text{sid})$ to the simulated $\mathcal{F}_{\text{STX}}$
  3. If the current mini-round is an update mini-round, then execute EXTENDLEDGERSTATE
- Upon receiving $(\text{FETCH-NEW}, \text{sid})$ from $\mathcal{A}$ (on behalf of a corrupted P) forward the request to the simulated $\mathcal{F}_{\text{STX}}$ and return whatever is returned to $\mathcal{A}$.
- Upon receiving $(\text{SEND}, \text{sid}, s, P')$ from $\mathcal{A}$ on behalf some *corrupted* party P, do the following:
  1. Forward the request to the simulated $\mathcal{F}_{\text{STX}}$.
  2. If the current mini-round is an update mini-round, then execute EXTENDLEDGERSTATE
  3. Return to $\mathcal{A}$ the return value from $\mathcal{F}_{\text{STX}}$.
- Upon receiving $(\text{SWAP}, \text{sid}, \text{mid}, \text{mid}')$ from $\mathcal{A}$, forward the request to the simulated $\mathcal{F}_{\text{STX}}$ and return whatever is returned to $\mathcal{A}$.
- Upon receiving $(\text{DELAY}, \text{sid}, T, \text{mid})$ from $\mathcal{A}$ forward the request to the simulated $\mathcal{F}_{\text{STX}}$ and do the following:
  1. Query the ledger state `state`
  2. Execute ADJUSTVIEW(`state`)
  3. Return to $\mathcal{A}$ the output of $\mathcal{F}_{\text{STX}}$

Simulation of the Network (over which transactionss are sent) :

- Upon receiving $(\text{MULTICAST}, \text{sid}, (m_{i_1}, P_{i_1}), \ldots, (m_{i_\ell}, P_{i_\ell}))$ with list of transactions from $\mathcal{A}$ on behalf some *corrupted* $P \in \mathcal{P}_{net}$, then do the following:
  1. Submit the transactions to the ledger on behalf of this corrupted party, and receive for each transaction the transaction id txid
  2. Forward the request to the internally simulated $\mathcal{F}_{\text{N-MC}}$, which replies for each message with a message-ID mid
  3. Remember the association between each mid and the corresponding txid
  4. Provide $\mathcal{A}$ with whatever the network outputs.

- Upon receiving (an ordinary input) $(\text{MULTICAST}, \text{sid}, m)$ from $\mathcal{A}$ on behalf of some *corrupted* $P \in \mathcal{P}_{net}$, then execute the corresponding steps 1. to 4. as above.
- Upon receiving $(\text{FETCH}, \text{sid})$ from $\mathcal{A}$ on behalf some *corrupted* $P \in \mathcal{P}_{net}$ forward the request to the simulated $\mathcal{F}_{\text{N-MC}}$ and return whatever is returned to $\mathcal{A}$.
- Upon receiving $(\text{DELAYS}, \text{sid}, (T_{\text{mid}_{i_1}}, \text{mid}_{i_1}), \ldots, (T_{\text{mid}_{i_\ell}}, \text{mid}_{i_\ell}))$ from $\mathcal{A}$ forward the request to the simulated $\mathcal{F}_{\text{N-MC}}$ and return whatever is returned to $\mathcal{A}$.
- Upon receiving $(\text{SWAP}, \text{sid}, \text{mid}, \text{mid}')$ from $\mathcal{A}$ forward the request to the simulated $\mathcal{F}_{\text{N-MC}}$ and return whatever is returned to $\mathcal{A}$.

procedure SIMULATEMINING(P, τ)

Simulate the mining procedure of P of the protocol:
if time-tick τ corresponds to a working sub-round **then**
Execute Step 2 of the mining protocol. This includes:
-Define the next state block $\mathtt{st}$ using the transaction set Txs_P
-Send $(\text{SUBMIT-NEW}, \text{sid}, \vec{\mathtt{st}}_P, \mathtt{st})$ to simulated functionality $\mathcal{F}_{\text{StX}}$.
-If successful, store $\vec{\mathtt{st}}_P || \mathtt{st}$ as the new $\vec{\mathtt{st}}_P$
-If successful, distribute the new state via $\mathcal{F}_{\text{StX}}$.
else if time-tick τ corresponds to an update sub-round **then**
Execute Step 4 of the mining protocol. This means that if the new information has not been fetched in this round already, then the following is executed:
-Fetch transactions $(\mathtt{tx}_1, \ldots, \mathtt{tx}_u)$ (on behalf of P) from simulated $\mathcal{F}_{\text{N-MC}}$ and add them to Txs_P.
-Fetch states $\vec{\mathtt{st}}_1, \ldots, \vec{\mathtt{st}}_s$ (on behalf of P) from the simulated $\mathcal{F}_{\text{StX}}$ and update $\vec{\mathtt{st}}_P$ to the largest state among $\vec{\mathtt{st}}_P$ and $\vec{\mathtt{st}}_i$.

procedure EXTENDLEDGERSTATE

Consider all honest and synchronized players P:
- Let $\vec{\mathtt{st}}$ be the longest state among all states $\vec{\mathtt{st}}_P$ or states contained in a receiver buffer $\vec{M}_P$ with delay 1 (and hence is a potential output in the next round)
Compare $\vec{\mathtt{st}}^{\lceil T}$ with the current state $\mathtt{state}$ of the ledger
if $|\mathtt{state}| > |\vec{\mathtt{st}}^{\lceil T}|$ **then**
Execute ADJUSTVIIEW($\mathtt{state}$)
if $\mathtt{state}$ is not a prefix of $\vec{\mathtt{st}}^{\lceil T}$ **then**
Abort the simulation (due to inconsistency)
Define the difference diff to be the block sequence s.t. $\mathtt{state}||\mathsf{diff} = \vec{\mathtt{st}}^{\lceil T}$.
Let $n \leftarrow |\mathsf{diff}|$
for each block diff_j, $j = 1$ to n **do**
Map each transaction $\mathtt{tx}$ in this block to its unique transaction ID txid
If a transaction does not yet have an txid, then submit it to the ledger and receive the corresponding txid from $\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$
Let $\mathsf{list}_j = (\text{txid}_{j,1}, \ldots, \text{txid}_{j,\ell_j})$ be the corresponding list for this block.
if coinbase $\text{txid}_{j,1}$ specifies a party that was honest at block creation time **then**

$\mathrm{hFlag}_j \leftarrow 1$
else
$\mathrm{hFlag}_j \leftarrow 0$
Output $(\textsc{next-block}, \mathrm{hFlag}_j, \mathsf{list}_j)$ to $\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$ (receiving $(\textsc{next-block}, ok)$ as an immediate answer)
Execute AdjustView($\mathtt{state}||\mathsf{diff}$)

procedure AdjustView($\mathtt{state}$)
$\mathsf{pointers} \leftarrow \varepsilon$
for each honest and synchronized party P_i **do**
Using the simulated functionality $\mathcal{F}_{\text{STX}}$ do the following:
- Let $\vec{\mathtt{st}}$ be the longest state among $\vec{\mathtt{st}}_{P_i}$ and those contained in the receiver buffer $\vec{M}_{P_i}$ with delay 1
Determine the pointer $\mathtt{pt}_i$ s.t. $\vec{\mathtt{st}}^{\lceil T} = \mathsf{state}|_{\mathtt{pt}_i}$
if such a pointer value does not exist **then**
Abort simulation (due to inconsistency)
if Party P_i has not executed step 4 of the mining protocol in this current mini-round **then**
$\mathsf{pointers} \leftarrow \mathsf{pointers}||(P_i, \mathtt{pt}_i)$
▷ As otherwise, the new state is only fetched in the next round
Output $(\textsc{set-slack}, \mathsf{pointers})$ to $\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$
$\mathsf{pointers} \leftarrow \varepsilon$
$\mathsf{desyncStates} \leftarrow \varepsilon$
for each honest but de-synchronized party P_i **do**
Using the simulated functionality $\mathcal{F}_{\text{STX}}$ do the following:
- Let $\vec{\mathtt{st}}$ be the longest state among $\vec{\mathtt{st}}_{P_i}$ and those contained in the receiver buffer $\vec{M}_{P_i}$ with delay 1
if Party P_i has not executed step 4 of the mining protocol in this current mini-round **then**
Set the pointer $\mathtt{pt}_i$ to be $|\vec{\mathtt{st}}^{\lceil T}|$
$\mathsf{pointers} \leftarrow \mathsf{pointers}||(P_i, \mathtt{pt}_i)$
$\mathsf{desyncStates} \leftarrow \mathsf{desyncState}||(P_i, \vec{\mathtt{st}}^{\lceil T})$
▷ As otherwise, the new state is only fetched in the next round
Output $(\textsc{set-slack}, \mathsf{pointers})$ to $\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$
Output $(\textsc{desync-state}, \mathsf{desyncStates})$ to $\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$

Bibliography

[ABC+07] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 598–609, New York, NY, USA, 2007. ACM.

[ABC+11] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14(1):12:1–12:34, June 2011.

[ACDV14] Elli Androulaki, Christian Cachin, Dan Dobre, and Marko Vukolić. Erasure-coded byzantine storage with separate metadata. In Marcos K. Aguilera, Leonardo Querzoni, and Marc Shapiro, editors, *Principles of Distributed Systems*, pages 76–90, Cham, 2014. Springer International Publishing.

[AD15] Marcin Andrychowicz and Stefan Dziembowski. Pow-based distributed cryptography with no trusted setup. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 379–399, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[ADDV16] Giuseppe Ateniese, Özgür Dagdelen, Ivan Damgård, and Daniele Venturi. Entangled cloud storage. *Future Gener. Comput. Syst.*, 62(C):104–118, September 2016.

[ADMM14] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Fair two-party computations via bitcoin deposits. In Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *Financial Cryptography and Data Security*, pages 105–121, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[ADMM16] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. *Commun. ACM*, 59(4):76–84, March 2016.

[ADPMT08] Giuseppe Ateniese, Roberto Di Pietro, Luigi V. Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, SecureComm '08, pages 9:1–9:10, New York, NY, USA, 2008. ACM.

[ADR02] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars R. Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002*, pages 83–107, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[AKST14] Daniel Apon, Jonathan Katz, Elaine Shi, and Aishwarya Thiruvengadam. Verifiable oblivious storage. In Hugo Krawczyk, editor, *Public-Key Cryptography – PKC 2014*, pages 131–148, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[An01] Jee Hea An. Authenticated encryption in the public-key setting: Security notions and analyses. Cryptology ePrint Archive, Report 2001/079, 2001. `https://eprint.iacr.org/2001/079`.

[BBM18] Christian Badertscher, Fabio Banfi, and Ueli Maurer. A constructive perspective on signcryption security. In B A, editor, *Security and Cryptography for Networks*, pages 0–0, Cham, 2018. Springer International Publishing.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 1–15, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[BD06] Tor E. Bjørstad and Alexander W. Dent. Building better signcryption schemes with tag-kems. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006*, pages 491–507, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BDOZ11] Moshe Babaioff, Shahar Dobzinski, Sigal Oren, and Aviv Zohar. On bitcoin and red balloons. *SIGecom Exch.*, 10(3):5–9, December 2011.

[BEG+94] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12(2):225–244, Sep 1994.

[BF08] Manuel Barbosa and Pooya Farshim. Certificateless signcryption. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2008, Tokyo, Japan, March 18-20, 2008*, pages 369–372. ACM, 2008.

[BF11] Paul Baecher and Marc Fischlin. Random oracle reducibility. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 21–38, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[BGK+18] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. Cryptology ePrint Archive, Report 2018/378, 2018. `https://eprint.iacr.org/2018/378`.

[BGM+18] Christian Badertscher, Juan Garay, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. But why does it work? a rational protocol design treatment of bitcoin. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology*

– *EUROCRYPT 2018*, pages 34–65, Cham, 2018. Springer International Publishing.

[BH03] Michael Backes and Dennis Hofheinz. How to break and repair a universally composable signature functionality. In Kan Zhang and Yuliang Zheng, editors, *Information Security*, volume 3225 of *LNCS*, pages 61–72, Heidelberg, 2003. Springer.

[BHMQU05] Michael Backes, Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. On fairness in simulatability-based cryptographic systems. In *Proceedings of the 2005 ACM Workshop on Formal Methods in Security Engineering*, FMSE '05, pages 13–22, New York, NY, USA, 2005. ACM.

[BJO09] Kevin D. Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: Theory and implementation. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 43–54, New York, NY, USA, 2009. ACM.

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, pages 421–439, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[BKR13] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Dupless: Server-aided encryption for deduplicated storage. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 179–194, Berkeley, CA, USA, 2013. USENIX Association.

[BM18] Christian Badertscher and Ueli Maurer. Composable and robust outsourced storage. In Nigel P. Smart, editor, *Topics in Cryptology – CT-RSA 2018*, pages 354–373, Cham, 2018. Springer International Publishing.

[BMM+15a] Christian Badertscher, Christian Matt, Ueli Maurer, Phillip Rogaway, and Björn Tackmann. Augmented secure channels and the goal of the tls 1.3 record layer. In Man-Ho Au and

Atsuko Miyaji, editors, *Provable Security*, pages 85–104, Cham, 2015. Springer International Publishing.

[BMM+15b] Christian Badertscher, Christian Matt, Ueli Maurer, Phillip Rogaway, and Björn Tackmann. Robust authenticated encryption and the limits of symmetric cryptography. In Jens Groth, editor, *Cryptography and Coding*, pages 112–129, Cham, 2015. Springer International Publishing.

[BMT18] Christian Badertscher, Ueli Maurer, and Björn Tackmann. On composable security for digital signatures. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography – PKC 2018*, pages 494–523, Cham, 2018. Springer International Publishing.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 324–356, Cham, 2017. Springer International Publishing.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000*, pages 531–545, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[Boy03] Xavier Boyen. Multipurpose identity-based signcryption. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 383–399, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[BPW03] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015, January 2003.

[BR00] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000*, pages 317–330, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 409–426, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BRW04] Mihir Bellare, Phillip Rogaway, and David Wagner. The eax mode of operation. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, pages 389–407, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[BSZ07] Joonsang Baek, Ron Steinfeld, and Yuliang Zheng. Formal proofs for the security of signcryption. *Journal of Cryptology*, 20(2):203–235, Apr 2007.

[But13] Vitalik Buterin. A next-generation smart contract and decentralized application platform. White Paper on GitHub, 2013. `https://github.com/ethereum/wiki/wiki/White-Paper`.

[Cac11] Christian Cachin. Integrity and consistency for untrusted services. In Ivana Černá, Tibor Gyimóthy, Juraj Hromkovič, Keith Jefferey, Rastislav Královič, Marko Vukolić, and Stefan Wolf, editors, *SOFSEM 2011: Theory and Practice of Computer Science*, pages 1–14, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Can01a] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42Nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, pages 136–147, Washington, DC, USA, 2001. IEEE Computer Society.

[Can01b] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42Nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, pages 136–, Washington, DC, USA, 2001. IEEE Computer Society.

[Can04] Ran Canetti. Universally composable signature, certification and authentication. In *Proceedings of CSFW 2004*, 2004.

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-damgård revisited: How to construct a hash function. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, pages 430–448, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[CDPW07a] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *Theory of Cryptography*, pages 61–85, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[CDPW07b] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *Theory of Cryptography*, pages 61–85, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[CDSMW08] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Black-box construction of a non-malleable encryption scheme from any semantically secure one. In Ran Canetti, editor, *Theory of Cryptography*, volume 4948 of *LNCS*, pages 424–441, 2008.

[CDV14] Christian Cachin, Dan Dobre, and Marko Vukolić. Separating data and control: Asynchronous bft storage with $2t + 1$ data replicas. In Pascal Felber and Vijay Garg, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 1–17, Cham, 2014. Springer International Publishing.

[CEK+16] Jan Camenisch, Robert Enderlein, Stefan Krenn, Ralf Küsters, and Daniel Rausch. Universal composition with responsive environments. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology — ASIACRYPT 2016*, volume 10032 of *LNCS*, pages 807–840. Springer, 2016.

[CEM16] Jan Camenisch, Robert R. Enderlein, and Ueli Maurer. Memory erasability amplification. In Vassilis Zikas and Roberto De Prisco, editors, *Security and Cryptography for Networks*, pages 104–125, Cham, 2016. Springer International Publishing.

[CG09] Christian Cachin and Martin Geisler. Integrity protection for revision control. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Applied Cryptography and Network Security*, pages 382–399, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[CGHZ16] Sandro Coretti, Juan Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multi-party computation based on one-way functions. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 998–1021, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[CKO14] Nishanth Chandran, Bhavana Kanukurthi, and Rafail Ostrovsky. Locally updatable and locally decodable codes. In Yehuda Lindell, editor, *Theory of Cryptography*, pages 489–514, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[CKS09] C. Cachin, I. Keidar, and A. Shraer. Fail-aware untrusted storage. In *2009 IEEE/IFIP International Conference on Dependable Systems Networks*, pages 494–503, June 2009.

[CKW13] David Cash, Alptekin Küpçü, and Daniel Wichs. Dynamic proofs of retrievability via oblivious ram. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 279–295, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[CMT13] Sandro Coretti, Ueli Maurer, and Björn Tackmann. Constructing confidential channels from authenticated channels—public-key encryption revisited. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, pages 134–153, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[CP13] Kai-Min Chung and Rafael Pass. A simple oram. Technical report, DTIC Document, 2013.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *Advances in Cryptology —*

CRYPTO 2003, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2003.

[CSS07] Christian Cachin, Abhi Shelat, and Alexander Shraer. Efficient fork-linearizable access to untrusted shared memory. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '07, pages 129–138, New York, NY, USA, 2007. ACM.

[CSV16a] Ran Canetti, Daniel Shahaf, and Margarita Vald. Universally composable authentication and key-exchange with global pki. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *Public-Key Cryptography – PKC 2016*, pages 265–296, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[CSV16b] Ran Canetti, Daniel Shahaf, and Margarita Vald. Universally composable authentication and key-exchange with global pki. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *Public-Key Cryptography – PKC 2016*, pages 265–296, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[DDM15a] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Compact attribute-based encryption and signcryption for general circuits from multilinear maps. In Alex Biryukov and Vipul Goyal, editors, *Progress in Cryptology – INDOCRYPT 2015*, pages 3–24, Cham, 2015. Springer International Publishing.

[DDM15b] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Functional signcryption: Notion, construction, and applications. In Man-Ho Au and Atsuko Miyaji, editors, *Provable Security*, pages 268–288, Cham, 2015. Springer International Publishing.

[Den05] Alexander W. Dent. Hybrid signcryption schemes with insider security. In Colin Boyd and Juan Manuel González Nieto, editors, *Information Security and Privacy*, pages 253–266, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[DGHM13] Grégory Demay, Peter Gaži, Martin Hirt, and Ueli Maurer. Resource-restricted indifferentiability. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 664–683, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[DP09] Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[DvDF+16] Srinivas Devadas, Marten van Dijk, Christopher W. Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. Onion oram: A constant bandwidth blowup oblivious ram. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography*, pages 145–174, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[DVW09] Yevgeniy Dodis, Salil Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In Omer Reingold, editor, *Theory of Cryptography*, pages 109–127, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[DZ10] Alexander W Dent and Yuliang Zheng. *Practical signcryption*. Springer Science & Business Media, 2010.

[EKPT09] Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 213–222, New York, NY, USA, 2009. ACM.

[ES18] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM*, 61(7):95–102, June 2018.

[Eya15] I. Eyal. The miner's dilemma. In *2015 IEEE Symposium on Security and Privacy*, pages 89–103, May 2015.

[FGMP15] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. Data is a stream: Security of stream-based channels. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 545–564, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[FNR+15] Christopher Fletcher, Muhammad Naveed, Ling Ren, Elaine Shi, and Emil Stefanov. Bucket oram: single online roundtrip, constant bandwidth oblivious ram. Technical report, IACR Cryptology ePrint Archive, Report 2015, 1065, 2015.

[GD02] Virgil D. Gligor and Pompiliu Donescu. Fast encryption and authentication: Xcbc encryption and xecb authentication modes. In Mitsuru Matsui, editor, *Fast Software Encryption*, pages 92–108, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[GGH+13] Craig Gentry, Kenny A. Goldman, Shai Halevi, Charanjit Julta, Mariana Raykova, and Daniel Wichs. Optimizing oram and using it efficiently for secure computation. In Emiliano De Cristofaro and Matthew Wright, editors, *Privacy Enhancing Technologies*, pages 1–18, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[GHJR15] Craig Gentry, Shai Halevi, Charanjit Jutla, and Mariana Raykova. Private database access with he-over-oram architecture. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *Applied Cryptography and Network Security*, pages 172–191, Cham, 2015. Springer International Publishing.

[GK07] Kristian Gjøsteen and Lillian Kråkmo. Universally composable signcryption. In Javier Lopez, Pierangela Samarati, and Josep L. Ferrer, editors, *Public Key Infrastructure*,

pages 346–353, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 281–310, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[GKZ10] J. A. Garay, A. Kiayias, and H. Zhou. A framework for the sound specification of cryptographic tasks. In *2010 23rd IEEE Computer Security Foundations Symposium*, pages 277–289, July 2010.

[GM11] Michael T. Goodrich and Michael Mitzenmacher. Privacy-preserving access of outsourced data via oblivious ram simulation. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, Languages and Programming*, pages 576–587, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[GM18] François Gérard and Keno Merckx. Setla: Signature and encryption from lattices. Cryptology ePrint Archive, Report 2018/056, 2018. `https://eprint.iacr.org/2018/056`.

[GMP+08] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. Universally composable security analysis of tls. In Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, editors, *Provable Security*, pages 313–327, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[GMPY11] Juan A. Garay, Philip MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. *Journal of Cryptology*, 24(4):615–658, Oct 2011.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ron Rivest. A digital signature scheme secure against adaptive chosen message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[HCH+15] John Hughes, Scott Cantor, Jeff Hodges, Frederick Hirsch, Prateek Mishra, Rob Philpott, and Eve Maler. Profiles for the Security Assertions Markup Language (SAML). OASIS Standard, March 2015.

[HHPSP11] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 491–500, New York, NY, USA, 2011. ACM.

[HKR15] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption aez and the problem that it solves. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 15–44, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[HSD+05] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C. Mitchell. A modular correctness proof of ieee 802.11i and tls. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, pages 2–15, New York, NY, USA, 2005. ACM.

[JK07] Ari Juels and Burton S. Kaliski, Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 584–597, New York, NY, USA, 2007. ACM.

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of tls-dhe in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 273–293, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Jut01] Charanjit S. Jutla. Encryption modes with almost free message integrity. In Birgit Pfitzmann, editor, *Advances in*

Cryptology — EUROCRYPT 2001, pages 529–544, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[KB14] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 30–41, New York, NY, USA, 2014. ACM.

[KB16] Ranjit Kumaresan and Iddo Bentov. Amortizing secure computation with penalties. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 418–429, New York, NY, USA, 2016. ACM.

[KKKT16] Aggelos Kiayias, Elias Koutsoupias, Maria Kyropoulou, and Yiannis Tselekounis. Blockchain mining games. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, EC '16, pages 365–382, New York, NY, USA, 2016. ACM.

[KLO12] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in)security of hash-based oblivious ram and a new balancing scheme. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 143–156, Philadelphia, PA, USA, 2012. Society for Industrial and Applied Mathematics.

[KMB15] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 195–206, New York, NY, USA, 2015. ACM.

[KMO+15] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann, and Daniele Venturi. (de-)constructing tls 1.3. In Alex Biryukov and Vipul Goyal, editors, *Progress in Cryptology – INDOCRYPT 2015*, pages 85–102, Cham, 2015. Springer International Publishing.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *Theory of Cryptography*, pages 477–498, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the tls protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 429–448, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[KT08] Ralf Küsters and Max Tuengerthal. Joint state theorems for public-key encryption and digitial signature functionalities with local computation. In *In Proc. 21st IEEE Computer Security Foundations Symposium (CSF'08*, 2008.

[Kup10] Alptekin Kupcu. *Efficient Cryptography for the Next Generation Secure Cloud*. PhD thesis, 2010.

[KVV16] Ranjit Kumaresan, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Improvements to secure computation with penalties. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 406–417, New York, NY, USA, 2016. ACM.

[KY01] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Bruce Schneier, editors, *Fast Software Encryption*, pages 284–299, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 705–734, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International, Menlo Park, California, October 1979.

[Lam98] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[Lam02] Leslie Lamport. Paxos made simple, fast, and byzantine. In *Procedings of the 6th International Conference on Principles of Distributed Systems. OPODIS 2002, Reims, France, December 11-13, 2002*, page 7–9. Suger, Saint-Denis, rue Catulienne, France, 2002.

[LBZ11] Joseph K. Liu, Joonsang Baek, and Jianying Zhou. Online/offline identity-based signcryption revisited. In Xuejia Lai, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology*, pages 36–51, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[LKMS04] Jinyuan Li, Maxwell Krohn, David Mazières, and Dennis Shasha. Secure untrusted data repository (sundr). In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 9–9, Berkeley, CA, USA, 2004. USENIX Association.

[LQ03] B. Libert and J. J. Quisquater. A new identity based signcryption scheme from pairings. In *Proceedings 2003 IEEE Information Theory Workshop*, pages 155–158, 2003.

[LQ04] Benoît Libert and Jean-Jacques Quisquater. Efficient signcryption with key privacy from gap diffie-hellman groups. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *Public Key Cryptography – PKC 2004*, pages 187–200, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[LW16] Bin Liu and Bogdan Warinschi. Universally composable cryptographic role-based access control. In Liqun Chen

and Jinguang Han, editors, *Provable Security*, pages 61–80, Cham, 2016. Springer International Publishing.

[Mau02] Ueli Maurer. Indistinguishability of random systems. In Lars R. Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002*, pages 110–132, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[Mau12] Ueli Maurer. Constructive cryptography – a new paradigm for security definitions and proofs. In Sebastian Mödersheim and Catuscia Palamidessi, editors, *Theory of Security and Applications*, pages 33–56, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MGGR13] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411, May 2013.

[ML02] John Malone-Lee. Identity-based signcryption. Cryptology ePrint Archive, Report 2002/098, 2002. `https://eprint.iacr.org/2002/098`.

[MMB15] Tarik Moataz, Travis Mayberry, and Erik-Oliver Blass. Constant communication oram with small blocksize. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 862–873, New York, NY, USA, 2015. ACM.

[MR11] Ueli Maurer and Renato Renner. Abstract cryptography. In *Innovations in Computer Science*. Tsinghua University Press, 2011.

[MR16] Ueli Maurer and Renato Renner. From indifferentiability to constructive cryptography (and back). In Martin Hirt and Adam Smith, editors, *Theory of Cryptography*, pages 3–24, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[MRT12] Ueli Maurer, Andreas Rüedlinger, and Björn Tackmann. Confidentiality and integrity: A constructive perspective. In

Ronald Cramer, editor, *Theory of Cryptography*, pages 209–229, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MS02] David Mazières and Dennis Shasha. Building secure file systems out of byzantine storage. In *Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing*, PODC '02, pages 108–117, New York, NY, USA, 2002. ACM.

[MSW08] P. Morrissey, N. P. Smart, and B. Warinschi. A modular security analysis of the tls handshake protocol. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, pages 55–73, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. White Paper, 2008. `http://bitcoin.org/bitcoin.pdf`.

[NR09] Moni Naor and Guy N. Rothblum. The complexity of online memory checking. *J. ACM*, 56(1):2:1–2:46, February 2009.

[Pat05] Akshay Patil. On symbolic analysis of cryptographic protocols. Massachusetts Institute of Technology, 2005. `http://hdl.handle.net/1721.1/33331`.

[PPB14] Tapas Pandit, Sumit Kumar Pandey, and Rana Barua. Attribute-based signcryption : Signer privacy, strong unforgeability and ind-cca2 security in adaptive-predicates attack. In Sherman S. M. Chow, Joseph K. Liu, Lucas C. K. Hui, and Siu Ming Yiu, editors, *Provable Security*, pages 274–290, Cham, 2014. Springer International Publishing.

[PRS11] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the tls record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 372–389, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[PS17] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '17, page 315–324, New York, NY, USA, 2017. ACM.

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 643–673, Cham, 2017. Springer International Publishing.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 184–200. IEEE, 2001.

[Rab83] M. O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 403–409, Nov 1983.

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. Ocb: A block-cipher mode of operation for efficient authenticated encryption. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, CCS '01, pages 196–205, New York, NY, USA, 2001. ACM.

[RFY+13] L. Ren, C. W. Fletcher, X. Yu, M. van Dijk, and S. Devadas. Integrity verification for path oblivious-ram. In *2013 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, Sept 2013.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 98–107, New York, NY, USA, 2002. ACM.

[Ros12] Mike Rosulek. Must you know the code of f to securely compute f? In Reihaneh Safavi-Naini and Ran Canetti,

editors, *Advances in Cryptology – CRYPTO 2012*, pages 87–104, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 373–390, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[RSA78] Ron Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[RSS11] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 487–506, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[SCC+10] Alexander Shraer, Christian Cachin, Asaf Cidon, Idit Keidar, Yan Michalevsky, and Dani Shaket. Venus: Verification for untrusted cloud storage. In *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop*, CCSW '10, pages 19–30, New York, NY, USA, 2010. ACM.

[SCG+14] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, May 2014.

[SDS+18] Emil Stefanov, Marten Van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious ram protocol. *J. ACM*, 65(4):18:1–18:26, April 2018.

[SSP13] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 325–336, New York, NY, USA, 2013. ACM.

[SSS12] Emil Stefanov, Elaine Shi, and Dawn Xiaodong Song. Towards practical oblivious RAM. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[SSVPR10] S. Sharmila Deva Selvi, S. Sree Vivek, and C. Pandu Rangan. Identity based public verifiable signcryption scheme. In Swee-Huay Heng and Kaoru Kurosawa, editors, *Provable Security*, pages 244–260, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[SvDJO12] Emil Stefanov, Marten van Dijk, Ari Juels, and Alina Oprea. Iris: A scalable cloud file system with efficient integrity checks. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 229–238, New York, NY, USA, 2012. ACM.

[SVVR12] S. Sharmila Deva Selvi, S. Sree Vivek, Dhinakaran Vinayagamurthy, and C. Pandu Rangan. Id based signcryption scheme in standard model. In Tsuyoshi Takagi, Guilin Wang, Zhiguang Qin, Shaoquan Jiang, and Yong Yu, editors, *Provable Security*, pages 35–52, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[SW13] Hovav Shacham and Brent Waters. Compact proofs of retrievability. *Journal of Cryptology*, 26(3):442–483, Jul 2013.

[SZ00] Ron Steinfeld and Yuliang Zheng. A signcryption scheme based on integer factorization. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Josef Pieprzyk, Jennifer Seberry, and Eiji Okamoto, editors, *Information Security*, pages 308–322, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[SZ15] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security*, pages 507–527, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[TP14] Youliang Tian and Changgen Peng. Universally composable secure group communication. Cryptology ePrint Archive, Report 2014/647, 2014. https://eprint.iacr.org/2014/647.

[WHC+14] Xiao Shaun Wang, Yan Huang, T-H. Hubert Chan, Abhi Shelat, and Elaine Shi. Scoram: Oblivious ram for secure computation. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 191–202, New York, NY, USA, 2014. ACM.

[WMAS13] Yang Wang, Mark Manulis, Man Ho Au, and Willy Susilo. Relations among privacy notions for signcryption and key invisible "sign-then-encrypt". In Colin Boyd and Leonie Simpson, editors, *Information Security and Privacy*, pages 187–202, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[WS96] David Wagner and Bruce Schneier. Analysis of the ssl 3.0 protocol. In *Proceedings of the 2Nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, WOEC'96, pages 29–40, Berkeley, CA, USA, 1996. USENIX Association.

[Zhe97] Yuliang Zheng. Digital signcryption or how to achieve cost(signature & encryption) « cost(signature) + cost(encryption). In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 165–179, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[ZI98] Yuliang Zheng and Hideki Imai. How to construct efficient signcryption schemes on elliptic curves. *Information Processing Letters*, 68(5):227 – 233, 1998.

[Zoh15] Aviv Zohar. Bitcoin: Under the hood. *Commun. ACM*, 58(9):104–113, August 2015.