Universität des Saarlandes
Department of Computer Science
Chair of Prof. Dr. Birgit Pfitzmann

Special Aspects of Escrow-based E-Cash Systems

Master's Thesis

Lennart Meier
<meier@krypt.cs.uni-sb.de>

Advisers:
Prof. Dr. Birgit Pfitzmann   Ahmad-Reza Sadeghi

27th March 2000

Versicherung

gemäß § 23 Abs. 2 der Prüfungsordnung für den Diplom-Studiengang Informatik

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Saarbrücken, den 27. März 2000

Abstract

Electronic cash systems are one way to implement digital payments; e-cash schemes can offer user anonymity at the same level as paper-based cash. However, this anonymity can be misused for illegal purposes; therefore, some means to revoke the anonymity under specific circumstances seems desirable. Many “escrow-based” schemes have been proposed; the information needed for this is usually escrowed at a trusted third party. In this thesis, we’ll take a close look at various escrow-based e-cash schemes and investigate their security for all participants and the assumptions it is based on. The notion of “self-escrowing”, i.e. replacing the trusted third party by the user himself, will be addressed; finally, by combining some of the techniques presented, we construct an escrow-based e-cash scheme which offers strengthened security against forgery of coins.

Zusammenfassung

Elektronische Münzsysteme (“e-cash”) sind eine Möglichkeit zur Implementierung digitaler Zahlungen. Diese Systeme können die Anonymität der Benutzer im gleichen Umfang wie herkömmliches Papiergeld gewährleisten. Diese Anonymität könnte jedoch für kriminelle Zwecke missbraucht werden; darum ist eine Möglichkeit zu ihrer Aufhebung bei gegebenem Anlass wünschenswert. Es sind viele sogenannte “escrow”-basierte (engl. to escrow - hinterlegen) deanonymisierbare Systeme vorgeschlagen worden; in ihnen wird die Information, die zur Anonymitätsaufhebung dient, bei einer vertrauten dritten Partei hinterlegt. In dieser Diplomarbeit werden einige solche Systeme vorgestellt und ihre Sicherheit für alle teilnehmenden Parteien sowie die zugrundeliegenden Annahmen betrachtet. Weiterhin wird “self-escrowing”, d.h. die Ersetzung der vertrauten dritten Partei durch den Benutzer selbst, untersucht; schließlich wird unter Kombination einiger der dargestellten Ansätze ein deanonymisierbares E-cash-System konstruiert, welches erhöhte Sicherheit gegen Münzfälschung bietet.

Acknowledgments

First of all, thanks to Prof. Dr. Birgit Pfitzmann for letting me write my Master's thesis in the interesting field of cryptography.

Special thanks must go to Ahmad-Reza Sadeghi for his around-the-clock advising and helping me in getting familiar with many things I knew nothing about. I appreciated his assistance with the not always easily digestible literature (he knows what I mean) and his numerous corrections and suggestions.

Many thanks to my family and friends for continuous support and for making me remember that there's so much more to life than one's studies. Thanks to all the people who made and make my humble residence "Wohnheim D" (http://www.heim-d.uni-sb.de) the best place to live in Saarbrücken.

The following professors of the computer science department helped me and other students to speed up our studies by offering additional courses beyond their line of duty: Prof. Dr. Wolfgang J. Paul, Prof. Dr. Birgit Pfitzmann, Prof. Dr. Gerhard Weikum. Thank you.

Last not least a big "thanks" to the authors of all the software used to typeset this thesis, namely LaTeX, BibTeX, MakeIndex, xfig, and other software running under Linux. Thanks also go to Mark A. Hillebrand for useful hints on using that software.

Contents

Chapter 1

Introduction

Electronic money, or electronic cash (e-cash), is getting more and more important. It can not only do away with some drawbacks of conventional money (size and weight, no loss-tolerance, time spent counting it), but some applications wouldn't even be possible without e-cash.

When buying goods in the "real world", instead of conventional cash the user has an "electronic wallet" (with his e-cash stored in it) used to pay arbitrary amounts in shops, at vending machines, etc. It is in electronic commerce, i.e. buying goods not at a real-world shop, but at a virtual one (e.g. in the World Wide Web), that e-cash is indispensable. Physical cash can't be used there for payment, as it can't be "sent" anywhere as digital data. Credit card transactions where only the credit card number and validity period are transmitted, as currently used by some shops in the WWW, are no option in the long term; they are insecure and can't be used when paying small amounts.

E-cash can be extended to offer various benefits that conventional cash simply can't provide, e.g. loss-tolerance. These extensions will not be further investigated here.

Ideally, e-cash should be easy to use and secure for all participants. This means that no party wants to suffer damage, material (i.e. financial) or immaterial (e.g. being falsely accused of a crime). Furthermore, anonymity for users, i.e. privacy of their payments (at least at the same level as with paper-cash) is desirable; several "coin-based" implementations providing user anonymity have been presented over the past years. However, anonymity can be misused for criminal activities. Therefore, a possibility to deanonymize transactions under specific circumstances is often required. In the systems presented so far, the information needed for deanonymization is "escrowed", i.e. deposited somewhere (e.g. at a trusted third party) from where it can be recovered if certain conditions are met, e.g. with a court order.

This Master's thesis is mainly concerned with security aspects of escrow-based e-cash systems. Chapter 2 provides the mathematical and cryptographical foundation. Topics covered include: problems from number theory such as computing discrete logarithms and factoring large integers; public-key cryptography and digital signatures; zero-knowledge proofs of knowledge (sic!).

In Chapter 3 the road that leads to escrow-based e-cash is presented. Beginning with a simple e-cash system, more and more features are added until we get to the point where escrowing is needed.

Chapter 4 is dedicated to escrowing with a trusted third party, the trustee. A few systems presented so far are briefly introduced and their security is discussed.

The aspect of self-escrowing is addressed in Chapter 5; some former results are presented and a general approach for transforming existing, trustee escrow-based e-cash schemes into self-escrow-based systems is investigated.

Chapter 6 deals with provable security of e-cash systems against coin forgery. The provability is in the random oracle model. Based on a secure signature scheme, an escrow-based e-cash system is constructed which is secure (in the random oracle model) against coin forgery.

In the last chapter we'll take a look back on the topics discussed in the previous chapters and give an outlook on open problems and alternative approaches to anonymity control not dealt with in this thesis.

Chapter 2

Mathematical and Cryptographical Background

This chapter provides an introduction into some mathematical and cryptographical topics that will be used in the following chapters. Readers familiar with these issues may skip this chapter.

2.1 Notation and Conventions

The notation and the conventions presented in this section will be used throughout this Master's thesis.

$\{0,1\}^*$ denotes the set of all binary strings. $a||b$ denotes the concatenation of (binary) strings a and b. $|a|_2$ is the length of the binary string a or of the binary representation of a value a. $x \in_R X$ means that x is chosen in random uniform manner from the set X. For any set S, $|S|$ denotes the number of elements in the set. For $a, b \in \mathbf{N}$, $\gcd(a, b)$ denotes the **g**reatest **c**ommon **d**ivisor of a and b.

When clear from the context, some conversions (e.g. from a value to a bit-string representing this value) and similar information might be omitted for reasons of clarity.

When speaking about bank customers who hold bank accounts, the terms customer "identity" and "account number" are regarded as equivalent, since an account number is always associated with at least one identity and vice versa. For simplicity, all systems are assumed to permit each user to have at most one bank account at a single bank (this is common practice and lets us use the equivalence described above). Moreover, each bank account belongs to exactly one user. Extensions to multiple accounts and accounts shared among users are straightforward.

2.1.1 Names

E-cash systems involve interacting parties or "players"; in the following, denotations as $\mathcal{U}$ for a user or $\mathcal{B}$ for a bank will be used. When talking about interactive protocols in general, the participants are denoted by $\mathcal{P}_1$, $\mathcal{P}_2$, etc.

When presenting existing systems, specifically their protocols, the names of parties, variables and functions found in the original articles have been adopted.

2.1.2 Interactive Protocols

The parties involved in e-cash transactions interact by protocols; these protocols are presented as shown in Figure 2.1: the participants are listed in the top row with initial values held by them written below their names in brackets. In the column under a participant's name computations performed by him are listed, while communication between the participants is indicated by an arrow in the center column. The information transmitted is described on top of and sometimes also below the arrow. In the bottom row of the protocol, again in brackets, you'll find the "results" that each participant gains from the execution of the protocol. The extension to three or more participants is straightforward.

If a protocol includes verifications like

$$\text{"verify A"} \quad \text{or} \quad \text{"}a \overset{?}{=} b\text{"},$$

this means that the participant checks the condition and proceeds with the protocol execution only if the verification is successful. Otherwise, the protocol is stopped and all participants are informed.

When saying that a participant is "honest" or that he executes the protocol "correctly", we mean that he acts exactly according to the protocol. The protocol itself is "correct" if it always yields the expected results when executed by honest participants. We'll call a protocol "sound" if it can't be manipulated by a cheating participant.

Transferring these terms to the "material world" would mean the following: A correct automatic teller machine allows honest users to withdraw money, i.e. no request from an honest user will be rejected. If the machine is also sound, it will allow money withdrawal only when all necessary requirements are met.

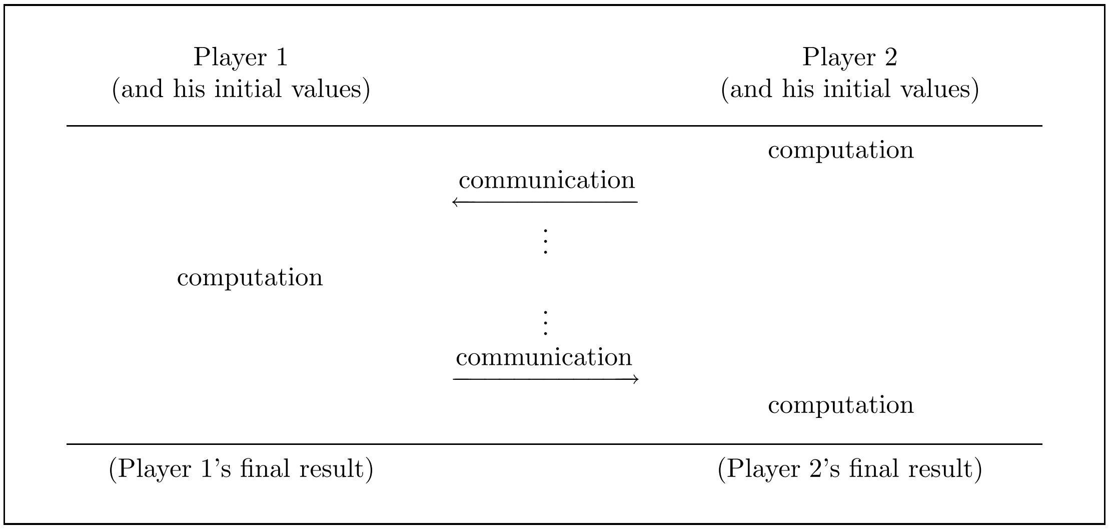


Figure 2.1: Example of an interactive protocol.

Definition 2.1 (View). The *view* of a participant in an interactive protocol is defined as the entire set of information the participant "sees" during the execution of the protocol.

Formally, the participants can be defined as Turing machines that share a common communication tape; it is the only tape both machines have access to. Thus, they can't exchange information by any other means. In this formal model, the view of a participant consists of the contents of the communication tape and of his "private" tapes.

In the protocol in Figure 2.1, the view of Player 1 would consist of all the values in the left and center column; likewise, the center and right column would constitute the view of Player 2.

2.2 Some Algebra

Most e-cash schemes are based on public-key cryptography. As the protocols used rely on specific mathematical properties of the algebraic groups the algorithms are applied in, this section gives a brief overview of the relevant algebraic foundations.

2.2.1 Groups

Definition 2.2 (Group). A *group* $(G, \circ)$ is a set G together with an associative binary operation $\circ$ such that

- $\circ$ is defined on all elements of $(G \times G)$ and returns an element of G
- there exists a unique identity element for $\circ$
- every element a of G has an inverse a^{-1} under $\circ$.

If $\circ$ is commutative, the group is called *Abelian* or commutative. $(G, \circ)$ is often denoted simply by G. For convenience, the multiplicative notation is frequently used; i.e. $ab = a \circ b$ and $a^x = a \circ a \circ \ldots \circ a$ (x times). For some $a \in G$, we denote by $\langle a \rangle$ the set of numbers obtained when raising a to the power of all $x \in \mathbf{Z}$, i.e. $\langle a \rangle := \{a^x | x \in \mathbf{Z}\}$. The set $\langle a \rangle$ is said to be *generated* by a.

Definition 2.3 (Cyclic group). A group G is called *cyclic* if there exists an element $g \in G$ such that every $a \in G$ can be written as g^x for some $x \in \mathbf{Z}$. g is called a *generator* of G, as $\langle g \rangle := \{g^x | x \in \mathbf{Z}\} = G$.

Definition 2.4 (Order of a group). The *order* of a finite group G is defined as the number of its elements, denoted by $|G|$.

Definition 2.5 (Order of a group element). The *order* of an element a of a finite group, denoted by $\mathrm{ord}(a)$, is the smallest positive integer x such that $a^x = 1$ (this implies that $|\langle a \rangle| = \mathrm{ord}(a)$).

Definition 2.6 (Subgroup). For a group G, a nonempty subset $H \subseteq G$ is called a *subgroup* of G if H is a group.

Theorem 2.1 (Lagrange's theorem on group order). For any group G and subgroup H of G the order of H divides the order of G [Zei96].

The above theorem implies that the order of any element of a finite group divides the order of the group (since the order of the element is equal to the order of the subgroup generated by it). If g is a generator of the cyclic group G of order m, then for $i \in \{1, \ldots, m\}$, $b = g^i$ has order $m / \gcd(m, i)$. Thus, b is a generator of G if and only if $\gcd(m, i) = 1$ (as for b to be a generator, $\mathrm{ord}(b)$ has to be m). Therefore, if m is prime, every element different from 1 is a generator of G.

2.2.2 The Groups $\mathbf{Z}_m$ and $\mathbf{Z}_m^*$

The set $\mathbf{Z}_m := \{0, \ldots, m-1\}$ together with addition modulo m constitutes a commutative group of order m. The set $\mathbf{Z}_m^* := \{a \in \mathbf{Z}_m | \gcd(m, a) = 1\}$, i.e. the set of integers between 1 and $m-1$ that are relatively prime to m, together with multiplication modulo m, forms another group.

Definition 2.7 (Euler ϕ-function). For a positive integer n, $\phi(n)$ is defined as the number of non-negative integers that are smaller than n and relatively prime to n:

$$\phi(n) := |\{a | 1 \leq a < n \text{ and } \gcd(a, n) = 1\}|.$$

The function ϕ is called the *Euler ϕ-function.*

$\mathbf{Z}_m$ is cyclic for all m (with 1 as a generator), whereas $\mathbf{Z}_m^*$ is cyclic if and only if its order $|\mathbf{Z}_m^*| = \phi(m)$ is 2, 4, or a power of an odd prime [MvOV97]. Groups of prime order, such as subgroups of $\mathbf{Z}_p^*$ for p prime, are often used in cryptography, as they have special properties (e.g. every element is a generator).

2.3 Problems from Number Theory

Most public-key cryptosystems are based on problems from number theory. In this section some of these problems are described; they will be instrumental in the schemes presented later on. Usually, each problem comes with a corresponding assumption, which consists in assuming that the problem is hard to solve; we'll give a notion of what "hard" means in Section 2.4. Below, the discrete logarithm problem is presented. The discrete logarithm assumption is that the discrete logarithm problem is hard to solve.

2.3.1 The Discrete Logarithm Problem

The difficulty of computing discrete logarithms is a problem many cryptosystems rely on. Therefore algorithms for solving this problem have been object of study [McC90, GM92].

Definition 2.8 (Discrete logarithm). Let G be a finite cyclic group with a generator g. The *discrete logarithm* of some element $a \in G$ with respect to g, denoted by $\log_g a$, is the unique integer x, $0 \leq x < |G|$, such that $a = g^x$.

Definition 2.9 (Discrete logarithm problem). The *discrete logarithm problem* is the following: given a finite cyclic group G, a generator g of it, and $a \in G$, find the discrete logarithm of a with respect to g.

For a cyclic group $G = \langle g \rangle$ of order m and $a, b \in G$ the following holds, analogously to ordinary logarithms:

- $\log_g(ab) \equiv \log_g(a) + \log_g(b) \pmod{m}$
- $\log_g(a^x) \equiv x \log_g(a) \pmod{m}$
- for $\langle h \rangle = G$ (i.e. h is another generator of G), $\log_g(a) \equiv log_h(a)(log_h(g))^{-1} \pmod{m}$

  NB. $(log_h(g))^{-1}$ is the multiplicative inverse, even if the group operation $\circ$ is the addition, i.e. $log_h(g)(log_h(g))^{-1} \equiv 1 \pmod{m}$ has to hold.

2.3.2 The Representation Problem

Definition 2.10 (Representation). Let G be a finite cyclic group of order m with distinct generators $g_1, \ldots, g_n \in G$. A *representation* of some element $a \in G$ with respect to $(g_1, \ldots, g_n)$ is an n-tuple $(x_1, \ldots, x_n), 0 \leq x_i < m$ for all $i \in \{1, \ldots, n\}$, such that

$$a = \prod_{i=1}^{n} g_i^{x_i}.$$

For any $a \in G$ there exist exactly m^{n-1} representations with respect to $(g_1, \ldots, g_n)$. If $a = 1$, the representation consisting of $(0, \ldots, 0)$ is called the trivial representation of a.

Definition 2.11 (Representation problem). Let G be a finite cyclic group of order m with distinct generators $g_1, \ldots, g_n \in G$. The *representation problem* is the following: given some element $a \in G$, find a representation of a with respect to $(g_1, \ldots, g_n)$.

The discrete logarithm problem is a special case of the representation problem (the number n of generators is 1). If the generators $g_1, \ldots, g_n$ are chosen randomly (i.e. uniformly and independently), finding two different representations of an element is as hard as solving the discrete logarithm problem.

2.3.3 The Diffie-Hellman Problem

Definition 2.12 (Diffie-Hellman problem). The *Diffie-Hellman problem* is the following: given a finite cyclic group G with a generator g, and values $g^u, g^v \in G$, find the element $g^{uv} \in G$.

The Diffie-Hellman problem is closely related to the discrete logarithm problem: if the latter can be solved in polynomial time, so can the Diffie-Hellman problem by first computing $u = \log_g(g^u)$ and then $(g^v)^u$.

Definition 2.13 (Decision Diffie-Hellman problem). The *Decision Diffie-Hellman problem* is the following: given a finite cyclic group G with a generator g, and values $g^u, g^v, g^w \in G$, decide whether $g^w = g^{uv}$.

Clearly, an algorithm that solves the Diffie-Hellman problem implicitly solves this one. For an extensive discussion of the Decision Diffie-Hellman problem, see [Bon98].

2.3.4 Factoring Large Integers

Definition 2.14 (Integer factorization problem). The *integer factorization problem* is the following: given a positive integer n, find its prime factorization, i.e. pairwise distinct primes p_i and positive integers e_i such that $n = \prod_{i=1}^{k} p_i^{e_i}$.

2.3.5 The RSA Problem

Definition 2.15 (RSA problem). The *RSA problem* is the following: given a group $\mathbf{Z}_n^*$ with $n = pq$ for two large primes p, q, a value $e \in \mathbf{Z}_{\phi(n)}^*$, and a value $a \in \mathbf{Z}_n^*$, find $b \in \mathbf{Z}_n$ such that $b^e = a$.

This is the problem the RSA cryptosystem [RSA78] is based on. It can be solved if $\phi(n) = (p-1)(q-1)$ is known. Thus, if the integer factorization problem is tractable, so is the RSA problem.

2.4 Public-Key Cryptography

Cryptosystems can be divided into two classes: symmetric systems and asymmetric ones. When considering an encrypted communication between two parties, the difference between symmetric and asymmetric encryption is the following:

- In a symmetric system, both parties share the same secret; technically speaking, encryption and decryption are done with the same secret key. For the communication to remain confidential, it is therefore crucial that this key doesn't become known to anyone else.
- In an asymmetric system, different keys are used for encryption and decryption. The encryption key is public (hence the term "public-key cryptography"), while the decryption key is known only to the receiver of the encrypted message.

Asymmetric cryptosystems offer several benefits over symmetric ones. In a symmetric cryptosystem, a different key is needed for each possible set of parties communicating with each other. Additionally, these keys have to be distributed confidentially, i.e. in a way that they don't become known to anyone else.

As the encryption key in an asymmetric system is public and used by anyone who wants to send encrypted messages to the holder of the corresponding decryption key, it can be distributed easily, e.g. by publishing it in a newspaper. The distribution is only required to be authentic, not confidential.

Public-key cryptography was first proposed by Diffie and Hellman in [DH76]. They introduced the notion of *trapdoor one-way functions.* These are functions that are easy to compute but hard to invert (i.e. one-way) unless one has some secret extra information (i.e. the "key" to the trapdoor). Given such a function f, a naive encryption can be implemented by publishing f as a public key, thus allowing anyone to encrypt messages m by computing $f(m)$. The decryption is done by computing $f^{-1}(f(m)) = m$, which is feasible only with the extra information mentioned above. NB. Not every one-way function f would hide and therefore encrypt the message m: given any one-way function f, define another function g as $g(x) := (f(x), y)$, where y is the first half of x. Then, g is also a one-way function, but certainly not suitable for encryption.

It should be well understood that inverting f is *hard, but not impossible.* The functions used in public-key cryptography are believed to be "hard enough" to invert; this means that, e.g., one can trust an encrypted message to remain confidential for a sufficient period of time. Roughly speaking, a given one-way function f is suitable if inverting f without the extra information takes "much longer" than computing f. As computers become faster and thus the time for inverting f gets shorter, f has to be replaced. In typical cryptosystems, f is some function $\tilde{f}(m, k)$ that takes not only the message m, but also the encryption key k as an argument. The time to compute f (and the time to invert it) grow with the size of k. Thus, it suffices to choose large enough keys (but without exaggerating, as this would result in an excessive need of time to compute f).

There are no proofs for the existence of one-way functions or a relation to assumptions like P $\neq$ NP. However, some functions are assumed to be one-way and the opposite hasn't been proven so far.

One-way functions are necessary and sufficient for digital signatures [Rom90], which are discussed in the following section. A digital signature should be a message-related value that can be computed only by the signer (with the secret key), but verified by anyone (using the public key). This resembles an "inverted" public-key cryptosystem, as the order in which the keys are used is reversed.

Analogously to the naive encryption described above, a naive signature on a message m could be $s = f^{-1}(m)$; anyone can check this signature by using the public key constituted by f: the signature is valid if $m = f(s)$. As for the encryption example, this should only give readers not familiar with these issues an impression of digital signatures. They are discussed more profoundly in Section 2.5.

2.4.1 Probabilistic Algorithms

The algorithms used in public-key cryptosystems are often probabilistic, i.e. there is a random value included in some of the computations. This is necessary, e.g., in key generation algorithms; a completely deterministic key generation algorithm would have to be kept secret to prevent others from using it to compute one's secret keys.

Also in the encryption example above, one would include some random value in the encryption of a message m. Like this, two encryptions of the same message would not be equal and therefore linkable anymore. The decryption would "throw away" the random values and thus yield m for both encryptions.

2.4.2 Hybrid Encryption

Asymmetric encryption schemes do have disadvantages with respect to symmetric ones: often, the message length is limited and the asymmetric systems are slower. Therefore, so-called *hybrid encryption* is frequently used: the actual message is encrypted with a symmetric system using a randomly chosen session key. Only this (relatively short) key is encrypted by public-key encryption.

2.4.3 Diffie-Hellman Key Exchange

In [DH76], a scheme for obtaining a common secret key, e.g. for hybrid encryption, using an authentic but not confidential channel was proposed. The scheme works as follows: Let G be a finite cyclic group of prime order q with a generator g such that computing discrete logarithms with respect to it is infeasible. Let x_1, x_2 be the secret keys of the two parties $\mathcal{P}_1$ and $\mathcal{P}_2$ that want to communicate with each other and $y_1 = g^{x_1}, y_2 = g^{x_2}$ the corresponding public keys, which are assumed to be distributed in an authentic manner. To obtain a common secret key k, $\mathcal{P}_1$ and $\mathcal{P}_2$ exchange their public keys and raise the partner's public key to the power of their own secret key, getting

$$k = y_1^{x_2} = y_2^{x_1} = g^{x_1 x_2}.$$

2.4.4 ElGamal Encryption

The following encryption scheme was proposed in [ElG85]: Let G be a finite cyclic group of order q with a generator g such that computing discrete logarithms is infeasible. To encrypt a message $m \in G$ for $\mathcal{P}_2$, $\mathcal{P}_1$ chooses $a \in_R \mathbf{Z}_q$ and computes the encryption of m as $(g^a, y^a m)$, where $y = g^x$ is the public key of $\mathcal{P}_2$. $\mathcal{P}_2$ can decrypt $(g^a, y^a m)$ using his secret key x:

$$\frac{y^a m}{(g^a)^x} = \frac{g^{xa} m}{g^{xa}} = m.$$

Alternatively, the encryption could be computed as $(y^a, g^a m)$, which could be decrypted in the following way:

$$\frac{g^a m}{(y^a)^{x^{-1}}} = \frac{g^a m}{g^{xax^{-1}}} = m.$$

The two schemes are equally secure, and they both rely on the intractability of the Diffie-Hellman problem (and therefore on that of the discrete logarithm problem); both encryptions are probabilistic, as a is chosen randomly.

2.5 Digital Signatures

A digital signature is a message-related value that can be efficiently computed only by the signer (i.e. by use of his secret key), but verified by anyone (using the public key). This means that the signature is linked to the public key and the message.

Since the purpose of signatures is to affirm something in a way verifiable by and provable to others, both the signer and the verifier of a signature (i.e. the receiver of the signed message) have interest in that digital signatures be unforgeable. The signer doesn't want to be held responsible for messages never actually signed by him, while the receiver of a signed message wants to be sure that it was signed by the legitimate holder of the corresponding secret key and that any third party, e.g. a judge, will be convinced of this fact.

Successful attacks on digital signature systems are divided in three groups, depending on what kind of forgery they achieve:

total break: The attacker is able to sign arbitrary messages, either by obtaining the secret key of the signer or by use of some other algorithm.

selective forgery: The attacker is able to compute a signature on a particular given message or a class of messages.

existential forgery: The attacker is able to forge at least one signature; he might not have any control over the choice of the message.

The attacker may interact with the signer, e.g. give him arbitrary messages to sign. The attack is successful if the attacker obtains a signature on a message the signer never signed. Existential forgery might not seem too dangerous when the attacker has no control over the message that is signed. This is only true as long as the set of messages "making sense" is much smaller than the set of senseless messages and as long as signatures on senseless messages don't cause damage (e.g. by staining one's reputation as a business partner).

2.5.1 Hash Functions

In signature schemes, hash functions $\mathcal{H}$ are used to reduce messages of arbitrary length to bit-strings of fixed length; the actual signing algorithm is then applied to these bit-strings. (Like this, the signing algorithm need not to be defined for inputs of varying length, and the resulting signatures are also of fixed length.) Hence, a signature on a message m usually is a signature on $\mathcal{H}(m)$. For signatures to be unforgeable, one must at least require that computing the hash function's inverse $\mathcal{H}^{-1}$ is infeasible, as otherwise with a signature on m an attacker could find all m' with $\mathcal{H}(m') = \mathcal{H}(m)$; the signature would also apply to all these messages.

A hash function $\mathcal{H}$ is said to be

- *weakly collision-resistant* if for a given x it is hard to find an $x' \neq x$ such that $\mathcal{H}(x') = \mathcal{H}(x)$,
- *strongly collision-resistant* if it is hard to find a pair (x, x') with $x' \neq x$ such that $\mathcal{H}(x') = \mathcal{H}(x)$, where $\mathcal{H}$ is chosen at random from a family of hash functions,
- *one-way* if for a given c it is hard to find an x such that $\mathcal{H}(x) = c$.

Any hash function may be inverted by simple brute-force search; keeping this in mind, a minimum number of output bits is reasonable. Additionally, care has to be taken that the hash function doesn't allow to find collisions by some other, more efficient means.

2.5.2 The Random Oracle Model

As stated above, when arguing about the security of a signature scheme, the hash function has to be taken in consideration. There exist provably secure schemes, but this security is achieved at the cost of efficiency. In order to obtain security arguments while keeping the scheme efficient, the random oracle model was introduced in [BR93]; in this model, the hash function $\mathcal{H}$ is seen as an oracle, i.e. as a function which produces a truly random output for each new query. Of course, if the same query is asked twice, the same value is output. It is argued that proofs in the random oracle model prove security of the overall scheme provided that the hash function has no weakness.

2.5.3 ElGamal Signatures

The following signature scheme was proposed in [ElG85] (in a simpler version, without hashing), as was the public-key encryption scheme described in Subsection 2.4.4.

Definition 2.16 (ElGamal signature). Let G be a finite cyclic group of order q with a generator g such that computing discrete logarithms is infeasible. Let x be the signer's secret key and $y := g^x$ the corresponding public key, and let $\mathcal{H}$ be a collision-resistant one-way hash function that maps $\{0,1\}^*$ to $\mathbf{Z}_q$.

An *ElGamal signature* with respect to the public key $y := g^x$ on a message $m \in \{0,1\}^*$ is a pair $(u, s) \in (G \times \mathbf{Z}_q)$ such that the verification equation

$$g^{\mathcal{H}(m)} \stackrel{?}{=} y^u u^s$$

holds.

Such a signature can be computed as follows:

1. choose $r \in_R \mathbf{Z}_q^*$,
2. compute $u := g^r$,
3. compute $s := r^{-1}(\mathcal{H}(m) - xu) \pmod q$.

Then, $y^u u^s = g^{xu} g^{rs} = g^{xu+rs} = g^{xu+\mathcal{H}(m)-xu} = g^{\mathcal{H}(m)}$; the signature is correct.

If computing discrete logarithms is feasible, signatures in this scheme can clearly be forged; the equivalence of these problems hasn't been proven. The random value r has to be different for each signed message; otherwise an attacker could take two signatures (u, s), (u, s') for messages m, m' and compute

$$\begin{array}{rcl} s - s' &=& r^{-1}(\mathcal{H}(m) - \mathcal{H}(m')) \\ \Rightarrow \quad r &=& \big(\mathcal{H}(m) - \mathcal{H}(m')\big)/(s - s') \\ \text{and thus} \quad x &=& \dfrac{\mathcal{H}(m) - sr}{u}. \end{array}$$

Finally, without (as in [ElG85]) a hash function $\mathcal{H}$, signatures could be forged existentially:

1. choose some β with an existing inverse, i.e. such that $\gcd(\beta, q) = 1$,
2. choose $\alpha \in_R G$ and set $u := g^\alpha y^\beta$,
3. compute $s := -u\beta^{-1}$,
4. set $m := \alpha s$.

The verification equation holds: $y^u u^s = y^u (g^\alpha y^\beta)^{-u\beta^{-1}} = g^{-u\alpha\beta^{-1}} = g^{\alpha s} = g^m$.

2.5.4 Schnorr Signatures

The Schnorr signature scheme from [Sch91] is also based on the discrete logarithm problem. Compared to the ElGamal system, it provides shorter signatures.

Definition 2.17 (Schnorr signature). Let G be a finite cyclic group of prime order q with a generator g such that computing discrete logarithms is infeasible. Let $x \in \mathbf{Z}_q$ be the signer's secret key and $y = g^x$ the corresponding public key, and let $\mathcal{H}$ be a collision-resistant one-way hash function that maps $\{0,1\}^*$ to $\mathbf{Z}_q$.

A *Schnorr signature* on a message $m \in \{0,1\}^*$ is a pair $(c, s) \in (\mathbf{Z}_q \times \mathbf{Z}_q)$ such that the verification equation

$$c \overset{?}{=} \mathcal{H}(m||g^s y^c)$$

holds.

Such a signature can be computed as follows:

1. choose $r \in_R \mathbf{Z}_q^*$,

2. compute $c := \mathcal{H}(m||g^r)$,

3. compute $s := r - cx \pmod{q}$.

Then, $\mathcal{H}(m||g^s y^c) = \mathcal{H}(m||g^{s+xc}) = \mathcal{H}(m||g^{r-cx+xc}) = \mathcal{H}(m||g^r) = c$; the signature is correct.

2.5.5 Chaum-Pedersen Signatures

The Chaum-Pedersen signature scheme from [CP92] is also based on the discrete logarithm problem.

Definition 2.18 (Chaum-Pedersen signature). Let G be a finite cyclic group of prime order q with a generator g such that computing discrete logarithms is infeasible. Let $x, y = g^x$ be the signer's secret and public key.

A *Chaum-Pedersen signature* on a message $m \in G$ is a tuple $(z, a, b, c, r) \in (G^3 \times \mathbf{Z}_q^2)$ such that the verification equations

$$g^r \overset{?}{=} ah^c$$

$$m^r \overset{?}{=} bz^c$$

hold.

Such a signature can be created in an interactive protocol between the signer Σ and the verifier $\mathcal{V}$ as in Figure 2.2: Σ computes the "signature commitment" z; the rest of the protocol serves to prove to $\mathcal{V}$ that z is of the form m^x.

If both participants are honest, $\mathcal{V}$ will obtain a valid signature (z, a, b, c, r) on m, since

$$g^r = g^{w+cx} = g^w (g^x)^c = ah^c$$

and

$$m^r = m^{w+cx} = m^w (m^x)^c = bz^c.$$

The signature can be created noninteractively by letting Σ compute c as $c := \mathcal{H}(m||z||a||b)$, where $\mathcal{H}$ is a collision-resistant one-way hash function that maps $\{0,1\}^*$ to $\mathbf{Z}_q$.

$$\begin{array}{ccc}
\mathcal{V} & & \Sigma \\
(m,g,y) & & (g,x) \\ \hline
& & z := m^x \\
& & w \in_R \mathbf{Z}_q \\
& & a := g^w \\
& & b := m^w \\
& \xleftarrow{\quad z,a,b \quad} & \\
c \in_R \mathbf{Z}_q & & \\
& \xrightarrow{\quad c \quad} & \\
& & r := w + cx \\
& \xleftarrow{\quad r \quad} & \\
g^r \overset{?}{=} ah^c & & \\
m^r \overset{?}{=} bz^c & & \\ \hline
(z,a,b,c,r) & &
\end{array}$$

Figure 2.2: Interactive Chaum-Pedersen signature.

2.6 Blind Signatures

Blind signatures were introduced by Chaum in [Cha83]; they allow a verifier to obtain signatures from a signer without the signer seeing the message to be signed or the signature itself. A blind signature is often said to correspond to a hand-written signature with closed eyes. This means that the signer does not see what he is signing; the signature, however, being authentic, will be accepted by any third party. This analogy is not perfect, as a digital signature is always the result of a computation performed by the signer. In addition, this computation involves the message to be signed. Obviously, it's impossible to perform calculations with numbers one does not know. Therefore the signer inevitably sees the message he is to sign.

The solution to this dilemma is to not provide the signer with the actual message m, but with a transformed (*blinded*) version m' of it. The transformation has to be such that the signer won't be able to know m after seeing and signing m'. In addition, there must exist an inverse transformation which on input the signature s' on m' will output a valid signature s on the original message m. Necessarily, the whole process of signing is no longer a single computation that is performed by the signer alone, but it takes place in an interactive protocol between the two parties. The verifier usually performs the blinding with values randomly chosen by him, so-called blinding factors [1].

Obviously, blind signatures are required to be unforgeable; since the computation of the signature is in part performed by the verifier (and potential attacker), the definition of a successful forgery is different from the one for nonblind signatures. An attacker succeeds in forging a blind signature, if the number of blind signatures obtained is larger than the number of protocol executions with the signer.

The additional desired feature called "blindness" is the following: when given a set of views of protocol executions and the (unordered) set of message-signature pairs related to these protocol runs, it must be infeasible for the signer to link views to message-signature pairs.

Blindness comes in two strengths: *statistical* blindness means that the protocol views and message-signature pairs are completely unlinkable, while *computational* blindness only requires linking of views to message-signature pairs to be as hard as some computationally infeasible problem.

[1] The word "factor" isn't used in the strict mathematical sense here, but rather means "ingredient"; the operation a blinding factor is used in is not necessarily multiplication.

2.6.1 Blind Schnorr Signatures

In the following, a scheme for obtaining blind Schnorr signatures is presented; many e-cash systems use them as a building block. The scheme was first proposed in [Oka93].

As in Schnorr's signature scheme, let G be a finite cyclic group of prime order q with a generator g such that computing discrete logarithms is infeasible. Let $x, y = g^x$ be the signer's secret and public key, and let $\mathcal{H}$ be a collision-resistant one-way hash function that maps $\{0,1\}^*$ to $\mathbf{Z}_q$.

If both participants follow the protocol in Figure 2.3, (c, s) is a valid Schnorr signature on m:

$$\mathcal{H}(m||g^s y^c) = \mathcal{H}(m||g^{s'+\alpha} y^{c'+\beta}) = \mathcal{H}(m||g^{r-c'x+\alpha+c'x} y^\beta) = \mathcal{H}(m||t' g^\alpha y^\beta) = \mathcal{H}(m||t) = c.$$

$$\begin{array}{ccc}
\mathcal{V} & & \Sigma \\
(m, g, y) & & (g, x) \\
\hline
 & & r \in_R \mathbf{Z}_q^* \\
 & & t' := g^r \\
 & \xleftarrow{\qquad t' \qquad} & \\
\alpha, \beta \in_R \mathbf{Z}_q & & \\
t := t' g^\alpha y^\beta & & \\
c := \mathcal{H}(m||t) & & \\
c' := c - \beta \pmod q & & \\
 & \xrightarrow{\qquad c' \qquad} & \\
 & & s' := r - c'x \pmod q \\
 & \xleftarrow{\qquad s' \qquad} & \\
s := s' + \alpha \pmod q & & \\
c \overset{?}{=} \mathcal{H}(m||g^s y^c) & & \\
\hline
(c, s) & &
\end{array}$$

Figure 2.3: Blind Schnorr signature.

To show that the protocol is (perfectly) blind it suffices to show that for every possible view of the signer Σ and for every possible signature there exists exactly one suitable pair of blinding factors (α, β). Given a view consisting of r, t', c', s', and a signature (c, s) on a message m, the only possibility is

$$\begin{aligned}
\alpha &:= s - s' \pmod q \\
\beta &:= c - c' \pmod q
\end{aligned}$$

With these blinding factors, the verifier $\mathcal{V}$ would have computed

$$\begin{aligned}
t &:= t' g^\alpha y^\beta \\
c &:= \mathcal{H}(m||t).
\end{aligned}$$

It remains to show that $t = g^s y^c$:

$$t = t' g^\alpha y^\beta = g^{r+\alpha+x\beta} = g^{r+s-s'+x(c-c')} = g^{s+xc} g^{r-s'-xc'} = g^s y^c.$$ [2]

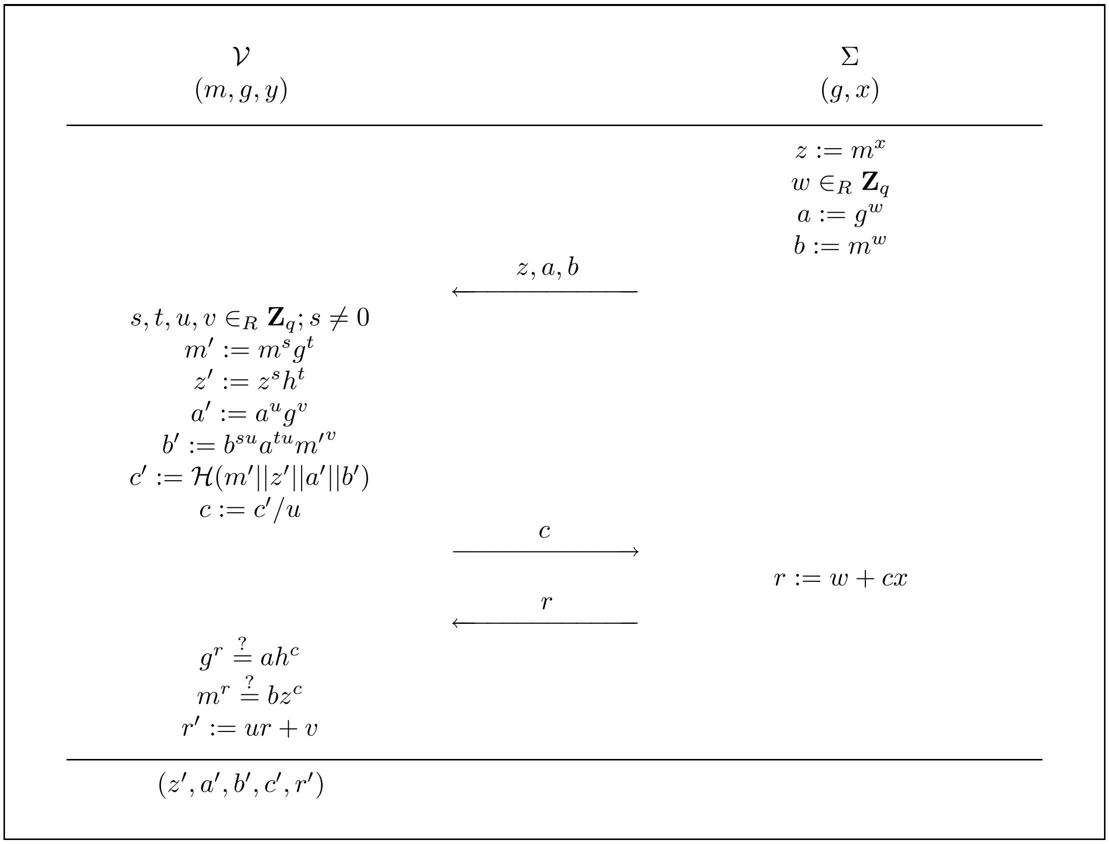
Thus, $c = \mathcal{H}(m||g^s y^c)$ and the blinding factors (α, β) would in fact have resulted in the valid signature (c, s).

[2] The last equality holds because $s' := r' - c'x \pmod q$.

2.6.2 Blind Chaum-Pedersen Signatures

The Chaum-Pedersen signature scheme from [CP92] exists also in a blind version.

As in the basic scheme, let G be a finite cyclic group of prime order q with a generator g such that computing discrete logarithms is infeasible. Let $x, y = g^x$ be the signer's secret and public key, and let $\mathcal{H}$ be a collision-resistant one-way hash function that maps $\{0,1\}^*$ to $\mathbf{Z}_q$.



$\mathcal{V}$ (m, g, y)		Σ (g, x)
		$z := m^x$ $w \in_R \mathbf{Z}_q$ $a := g^w$ $b := m^w$
	$\xleftarrow{z, a, b}$	
$s, t, u, v \in_R \mathbf{Z}_q; s \neq 0$ $m' := m^s g^t$ $z' := z^s h^t$ $a' := a^u g^v$ $b' := b^{su} a^{tu} m'^{v}$ $c' := \mathcal{H}(m' \vert\vert z' \vert\vert a' \vert\vert b')$ $c := c'/u$		
	$\xrightarrow{c}$	$r := w + cx$
	$\xleftarrow{r}$	
$g^r \overset{?}{=} ah^c$ $m^r \overset{?}{=} bz^c$ $r' := ur + v$		
(z', a', b', c', r')		

Figure 2.4: Blind Chaum-Pedersen signature.

If the verifier $\mathcal{V}$ and the signer follow the protocol shown in Figure 2.4, the verifier $\mathcal{V}$ will obtain a valid signature (z', a', b', c', r') on m', since

$$g^{r'} = g^{ur+v} = g^{uw+ucx} g^v = g^{wu} g^v g^{xc'} = a' h^{c'}$$

and

$$m'^{r'} = m'^{ur+v} = m'^{uw+ucx} m'^{v} = m'^{wu} m'^{v} m'^{xc'} = (m^s g^t)^{wu} m'^{v} z'^{c'} = m^{wsu} g^{wtu} m'^{v} z'^{c'} = b' z'^{c'}.$$

There is an additional verification equation

$$c' \overset{?}{=} \mathcal{H}(m'||z'||a'||b')$$

that obviously holds for signatures resulting from the correct execution of the protocol. The reason for this further verification is that without it, (z, a, b, c, r) would be a valid signature on m: as can be seen, the values z, a, b, r and z', a', b', r' are closely related; indeed, there's even a value $w' = wu + v$ such that $a' = g^{w'}$ and $b' = m'^{w'}$, in perfect analogy to the structure of a and b. However, with the additional verification equation, (z, a, b, c, r) is no valid signature on m, since $c := u \times \mathcal{H}(m'||z'||a'||b') \neq \mathcal{H}(m||z||a||b)$.

As with blind Schnorr signatures, the blindness of the protocol can be proven by showing that for every possible view of the signer and for every possible signature there exists exactly one suitable quadruple of blinding factors. The proof can be found in [Pfi99].

NB. The protocol can always be executed with $t = 0$, and blindness is guaranteed also in that case. We'll need this later on.

2.7 Proofs of Knowledge

A proof of knowledge is used to convince someone that one has knowledge of something, usually of a value. The easiest way would clearly be to share this knowledge, i.e. to show the value, but sometimes it is crucial that the value remain secret. One might, e.g., want to prove one's knowledge of the secret key related to a certain public encryption key; obviously proving this by showing the secret key is no valid option. Therefore, knowledge is often proven by proving one's ability to efficiently compute something. This is where *zero-knowledge* proofs of knowledge come into action.

2.7.1 Zero-Knowledge Property

Definition 2.19 (Zero knowledge). An interactive protocol is said to be *perfectly / statistically / computationally zero knowledge*, if for every probabilistic polynomial-time verifier $\mathcal{V}$ there exists a probabilistic expected polynomial-time simulator $\mathcal{S}_\mathcal{V}$ such that the outputs of $\mathcal{S}_\mathcal{V}$ and of the protocol when executed by $(\mathcal{V}, \mathcal{P})$ (with $\mathcal{P}$ the prover) are perfectly / statistically / computationally indistinguishable [Pfi98].

What this means is that the verifier shouldn't be able to gain any new knowledge from the execution of the interactive protocol with the prover. Therefore, he might compute his view of the protocol by himself, i.e. using the simulator. This is sketched in Figure 2.5. The inability to distinguish such a self-computed view from a real one is exactly what results from the zero-knowledge property of the protocol. Computational indistinguishability is often enough in cryptography, as the verifier is assumed to have limited computational power. In Figure 2.5 this means that the distinguisher is a probabilistic polynomial-time algorithm and as such can't distinguish a simulated view from a real one.

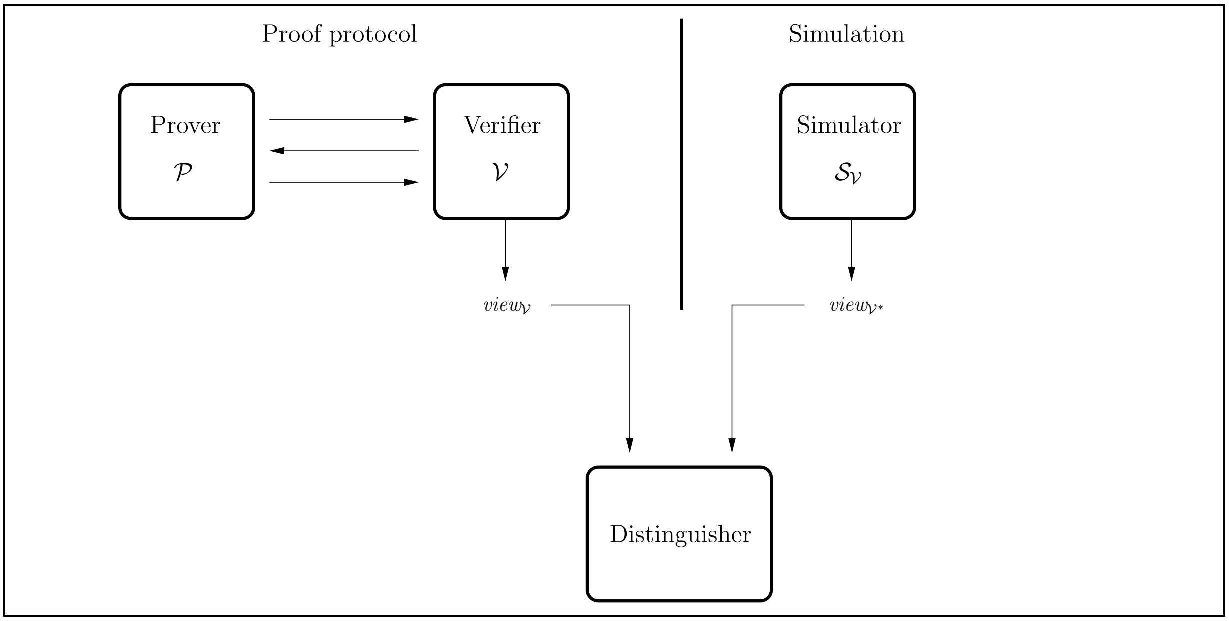


Figure 2.5: Indistinguishability of views.

2.7.2 Knowledge Extractors

To examine whether a knowledge proof actually proves knowledge, a so-called knowledge extractor $\mathcal{E}$ can be applied. $\mathcal{E}$ is defined as a probabilistic polynomial-time algorithm that acts as the verifier in the knowledge proof protocol and tries to obtain the prover's secrets. To this end, $\mathcal{E}$ has a special ability: it can "reset" the prover at any time and let him repeat specific steps of the protocol with other inputs. Thus, $\mathcal{E}$ can obtain information that a normal verifier would never see; it's for this reason that $\mathcal{E}$ can extract knowledge even from zero-knowledge protocols.

2.7.3 Proof of Knowledge of a Discrete Logarithm

A zero-knowledge scheme for proving knowledge of a discrete logarithm was presented in [CEvdG88]. The scheme is almost equal to the Schnorr identification protocol shown in Figure 2.6; the challenge c is a single bit (which makes the scheme perfect zero-knowledge), and the protocol is executed several times (which limits the knowledge error, i.e. the probability that the verifier $\mathcal{V}$ accepts although the prover $\mathcal{P}$ doesn't know the secret). For $|q|_2$ repetitions, both the zero-knowledge property and the soundness of the scheme can be proven [Pfi98].

Schnorr's identification protocol is much more efficient, as only one iteration is needed, but it is not believed to be zero-knowledge.

Using an idea from [FS86] (creating signature schemes from three-move identification protocols by replacing the random challenge c by $c := \mathcal{H}(m||a)$, the Schnorr identification protocol becomes the Schnorr signature scheme as seen in Subsection 2.5.4.

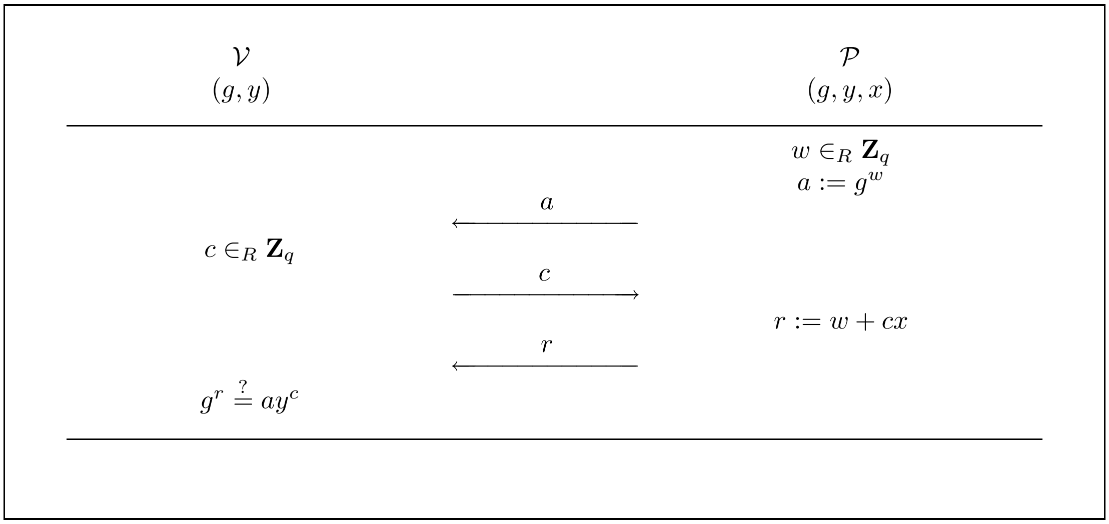


Figure 2.6: Schnorr identification protocol.

Different variations of proofs of knowledge of a discrete logarithm appear in the literature. In the notation of [CMS96], a noninteractive proof of knowledge of the discrete logarithm of a group element h with respect to a generator g consists of a Schnorr signature (c, s) related to a public key (g, h) on the message $m||g||h$. This proof is dependent on the message m, which may be the empty string ϵ. It is denoted by

$$PKLOG(m, g, h) = (c, s),$$

the verification equation is $c \overset{?}{=} \mathcal{H}(m||g||h||g^s h^c)$.

When using such proofs, we'll use clear denotations like $Proof_{KLOG}$ and in case of need remind the reader of the specific meaning.

2.7.4 Proof of Equality of Discrete Logarithms

A proof of equality of discrete logarithms is intended to prove equality of $\log_{g_1} h_1$ and $\log_{g_2} h_2$ for values g_1, h_1, g_2, h_2; in addition, it should prove knowledge of this logarithm. It is denoted as $Proof_{EQLOG}$. Like with proofs of knowledge as presented in the previous subsection, there are different proof protocols with different security and efficiency properties [CEvdG88].

The noninteractive proof from [CMS96] is an extension of a Schnorr signature and consists of a pair (c, s) satisfying the verification equation

$$c \stackrel{?}{=} \mathcal{H}(m||g_1||g_2||h_1||h_2||g_1^s h_1^c||g_2^s h_2^c).$$

The proof is dependent on the message m, which may be the empty string ϵ. It is denoted by

$$PLOGEQ(m, g_1, h_1, g_2, h_2) = (c, s)$$

and can be computed as follows:

1. choose $r \in_R \mathbf{Z}_q$,
2. compute $c := \mathcal{H}(m||g_1||g_2||h_1||h_2||g_1^r||g_2^r)$,
3. compute $s := r - cx \pmod{q}$.

Chapter 3

Electronic Cash Systems

The purpose of digital payment systems is to transfer money, typically in the context of purchasing (material or immaterial) goods: the buyer pays for goods delivered by the seller. There exist different types of payment models [BP89, AJSW97]; electronic cash, where payment is done with “coins”, represents one of them. Note that the term “electronic cash” is sometimes also used for digital payment systems that don’t employ coins.

Electronic cash systems can replace conventional paper-based cash, i.e. the buyer pays with an electronic wallet by letting it interact with some device of the seller. Additionally, they make electronic payments over long distances (e.g. via the World Wide Web) possible.

In this chapter, the road that leads to escrow-based e-cash is presented. Beginning with a simple digital payment system, more and more features considered useful are added until we get to the point where escrowing is required.

3.1 Simple Digital Payments

In a simple form, digital payments involving a user $\mathcal{U}$, a shop $\mathcal{S}$ and a bank $\mathcal{B}$ can be accomplished as in Figure 3.1. Merely some means of authentication is needed, i.e. there mustn’t be any doubt about $\mathcal{U}$’s identity and the action he wants to take. Digital signatures provide an efficient way to accomplish this.

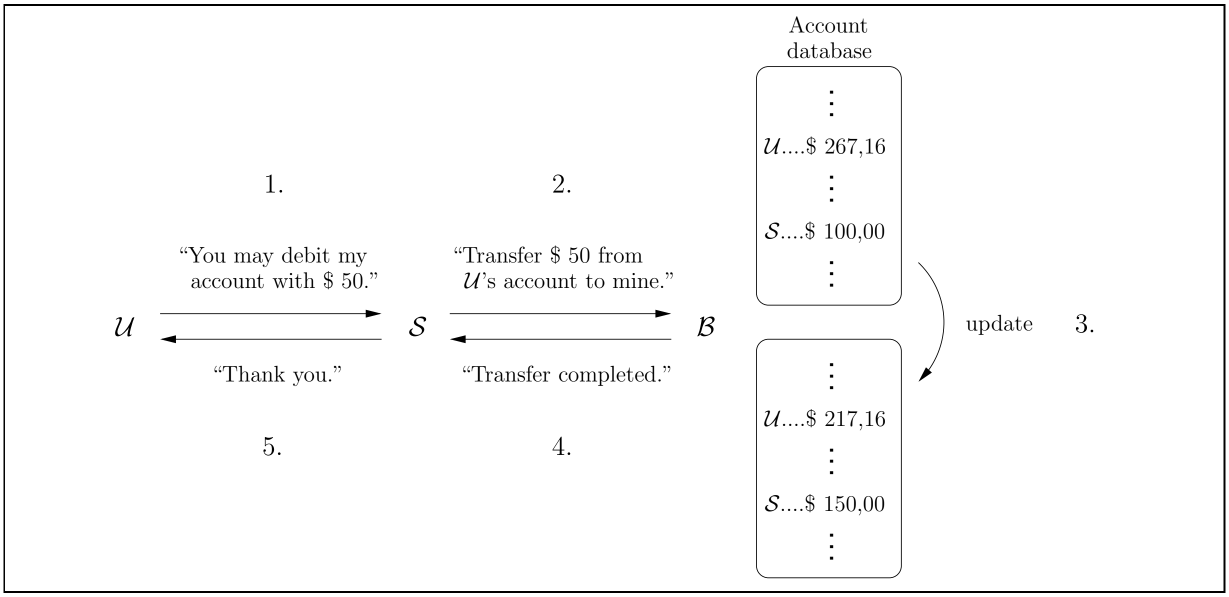

Account database
⋮
$\mathcal{U}$....\$ 267,16
⋮
$\mathcal{S}$....\$ 100,00
⋮
1.
2.
“You may debit my account with \$ 50.”
“Transfer \$ 50 from $\mathcal{U}$’s account to mine.”
$\mathcal{U}$
$\mathcal{S}$
$\mathcal{B}$
update
3.
“Thank you.”
“Transfer completed.”
⋮
$\mathcal{U}$....\$ 217,16
⋮
$\mathcal{S}$....\$ 150,00
⋮
5.
4.

Figure 3.1: Simple digital payment.

This simple system is exactly the one used when paying with a conventional credit card in a shop. $\mathcal{U}$ proves his identity by presenting the credit card (a cautious $\mathcal{S}$ might also require identity papers). $\mathcal{S}$ lets $\mathcal{B}$ check that the account has the required credit. To assure $\mathcal{B}$ that $\mathcal{U}$ agrees on his account being debited and that the credit card indeed belongs to $\mathcal{U}$, $\mathcal{S}$ usually provides $\mathcal{B}$ with a paper-based signature by $\mathcal{U}$ or $\mathcal{U}$ enters his secret PIN into a terminal which forwards it to $\mathcal{B}$.

Such a simple payment system has two disadvantages:

- need for on-line communication with $\mathcal{B}$ at each payment,
- almost total loss of anonymity for $\mathcal{U}$; $\mathcal{B}$ knows when and where $\mathcal{U}$ spends his money.

These drawbacks (and the fees charged by $\mathcal{B}$ for every transaction) have contributed to conventional credit cards being used only for payments above a certain amount. In the same way, they make simple electronic transactions as in Figure 3.1 for small amounts unattractive for all parties.

3.2 Security Requirements

In the previous section, anonymity for $\mathcal{U}$ was mentioned; this is not only a "desirable feature", but actually a requirement for $\mathcal{U}$'s security as a participant in the payment system. There are such security requirements of all participants [1]; as in everyday life, the general way to describe them all in one sentence is: "Nobody wants to suffer any damage." For being able to build systems that are "safe" for everyone, more precise definitions are needed. Apart from the abstract nature of universal statements as the one above, there are problems no digital payment system can solve (e.g. blackmailing of $\mathcal{B}$) or that don't even depend on the payment system itself (e.g. honesty of judges). Thus, having defined the security requirements, one also knows which aspects are intractable and isn't distracted by them anymore.

Another general requirement of all participants is: when behaving honestly, each of them wants to be safe from false accusations. This can generally be achieved by making the protocols such that the correct behavior can be proven by use of the protocol transcripts (which in turn have to be certified, e.g. by digital signatures).

This and other requirements are presented in the following for each participant in a digital payment system (presuming that there are no other participants).

Requirements of $\mathcal{U}$

$\mathcal{U}$ normally trusts $\mathcal{B}$ not trying to steal his money; otherwise, all transactions with $\mathcal{B}$ can be digitally signed by both, thus preventing disputes (at the cost of elevated computational effort and communication). Apart from this,

- $\mathcal{U}$ doesn't want to lose money; this means that nobody should be able to withdraw coins under his name or to steal and spend coins withdrawn by $\mathcal{U}$,
- $\mathcal{U}$ wants anonymity for his payments; nor $\mathcal{B}$ nor anyone else should be able to know how $\mathcal{U}$ spends his money. This information could be used to create user profiles or, even worse, to blackmail $\mathcal{U}$. We distinguish untraceability or "simple anonymity" and unlinkability (of payments). Untraceability means that user identities should be unlinkable to payments (i.e. nobody should be able to tell whether $\mathcal{U}$ performed a certain payment or not), while unlinkability of payments means that it is infeasible to know whether two payments were made by the same user. If payments are untraceable but not unlinkable, user profiles can be created and in the end, even $\mathcal{U}$'s anonymity might be compromised.
- $\mathcal{U}$ doesn't want to be falsely accused of overspending, i.e. spending a coin more than once,

[1] There are also systems that offer anonymity for the payment receiver; they aren't dealt with in this thesis.

- in case $\mathcal{U}$ indeed overspends, the corresponding payment transcripts shouldn't allow anyone else (including $\mathcal{B}$) to do further overspending of the same coin. This means that even an overspent coin should remain spendable only by $\mathcal{U}$, so that $\mathcal{U}$ won't be held liable for further overspending done by others.

Requirements of $\mathcal{B}$

The security requirement of $\mathcal{B}$ mainly taken in consideration in this thesis is that of not losing money (another might be the one of not losing credibility). Therefore,

- coins mustn't be forgeable; more precisely, it should be infeasible to obtain more valid coins than were actually issued by $\mathcal{B}$ in correct executions of the withdrawal protocol,
- each coin must be spendable only once; in case this can't be guaranteed, double-spenders must be identifiable once the fact has been detected. Double-depositing by $\mathcal{S}$ must also be prevented or at least detected afterwards.

Requirements of $\mathcal{S}$

The security for $\mathcal{S}$ relies mainly on the amount of legal responsibility $\mathcal{S}$ assumes in the whole system; typically, $\mathcal{B}$ bears much of the risks. If, e.g., $\mathcal{S}$ deposits a forged but valid coin, $\mathcal{B}$ would still have to accept it (because it can only be recognized as forged because it doesn't appear in $\mathcal{B}$'s list of withdrawn coins - but this list is no legal evidence, as $\mathcal{B}$ might have modified it). Apart from this,

- $\mathcal{S}$ doesn't want to be falsely accused of double-depositing a coin,
- theft or extortion of coins paid to $\mathcal{S}$ should be prevented by making such coins depositable only by $\mathcal{S}$, typically by including the identity of $\mathcal{S}$ in the payment transcript that is presented to $\mathcal{B}$ for deposit.

3.3 Coin-based E-Cash

In Figure 3.2, the main structure of coin-based digital payments or e-cash is presented. It involves (apart from system setup procedures such as, e.g., key generation and account opening) three distinct transactions:

- During *withdrawal* $\mathcal{B}$ issues an "electronic coin" c, typically consisting of a (random) number n and a signature $sig(sk_B, n)$ on this number, where sk_B denotes $\mathcal{B}$'s secret key.
- At *payment* $\mathcal{U}$ hands the coin to $\mathcal{S}$ who accepts it if $\mathcal{B}$'s signature on it is valid.
- $\mathcal{S}$' account is credited only at *deposit* time: $\mathcal{S}$ sends the coin to $\mathcal{B}$ who verifies the validity of the signature and checks that the coin hasn't already been deposited. If both verifications succeed, $\mathcal{S}$' account is credited with the value of the coin.

Each coin is agreed to have a certain value. Different coin denominations can be implemented by using different signing keys for different coin denominations.

If payment and deposit are executed simultaneously, we have a so-called *on-line* system (i.e. $\mathcal{B}$ is on-line at payment time) with no obvious benefits over the simple system presented in Section 3.1; actually the amount of communication between the parties is even higher.

If payment and deposit are executed separately, we obtain an *off-line* system. The advantage of off-line e-cash is that $\mathcal{B}$ hasn't to be contacted at each payment: $\mathcal{S}$ and $\mathcal{B}$ can interact for deposit at regular intervals, depositing a major number of coins in a single communication.

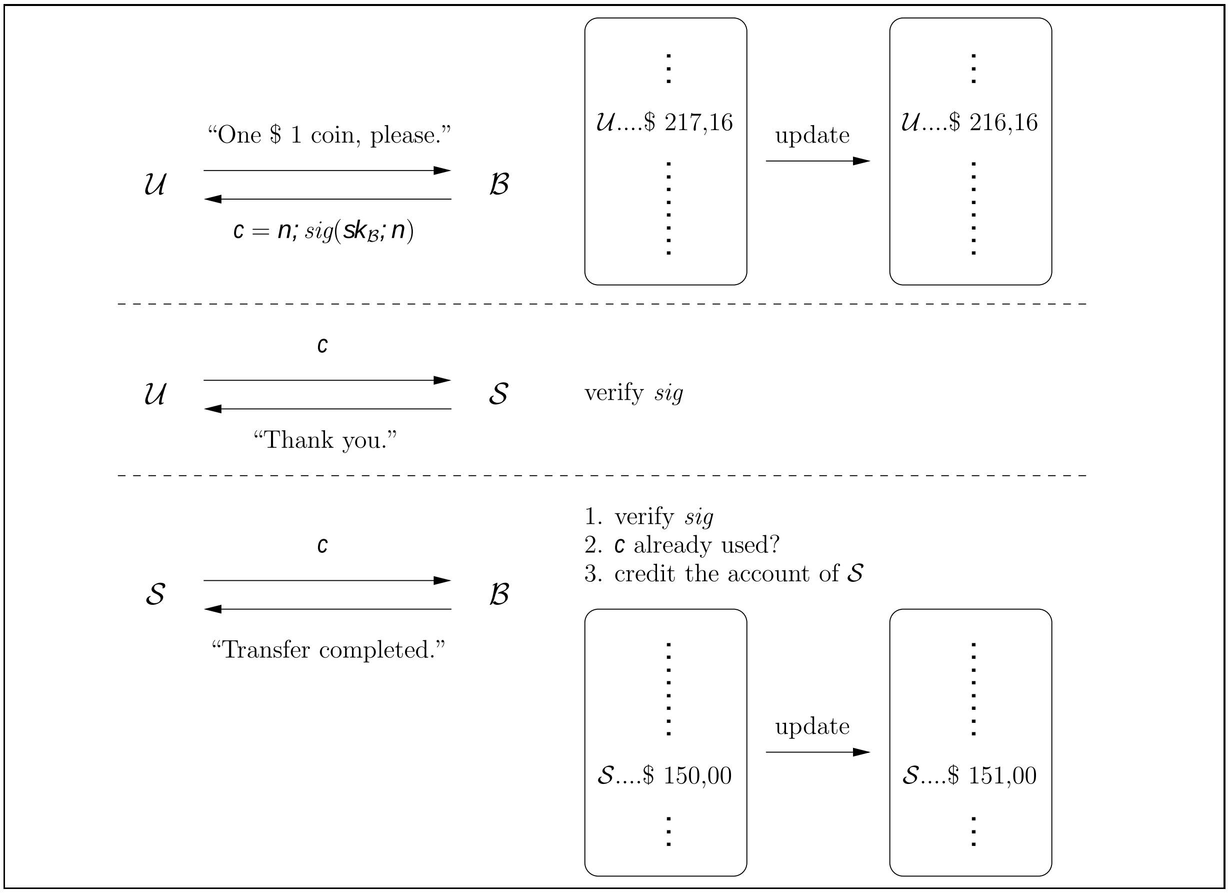


Figure 3.2: Off-line e-cash.

The elevated computational effort of e-cash with respect to the simple payment system is not excessively high, it might even not exist (depending on the complexity of the authentication). However, poor anonymity for $\mathcal{U}$ still remains a problem. Moreover, in such a simple off-line e-cash system anyone (thus, also $\mathcal{S}$) can (re)spend a coin, as coins are not yet spendable only by the withdrawer. In the following sections, ways to achieve anonymity and to prevent overspending are investigated.

3.4 Achieving User Anonymity

To achieve anonymity for $\mathcal{U}$, we should focus on how it is actually disrupted. The problem is that $\mathcal{B}$ sees the electronic coin at withdrawal, when $\mathcal{U}$ has to prove his identity $I_{\mathcal{U}}$ to $\mathcal{B}$. Therefore $\mathcal{B}$ can link the coin to $\mathcal{U}$ by simply adding (coin, $I_{\mathcal{U}}$) to an appropriate database. At deposit time, the coin is handed to $\mathcal{B}$ who may look up its withdrawer in the database.

The concept of *blind signatures* presented in Subsection 2.6 helps in solving this problem: if $\mathcal{B}$ issues the coin by use of a blind signature, he won't see the coin that $\mathcal{U}$ actually hands to $\mathcal{S}$ at payment time. When receiving the coin for deposit, linking it to the withdrawal is infeasible; thus, the anonymity of $\mathcal{U}$ should be retained.

3.5 Double-Spending

So far, we have achieved anonymous off-line e-cash using blind signatures. The anonymity of $\mathcal{U}$, however, immediately gives birth to a new problem inherent to electronic cash systems: Electronic coins are digital data and can therefore be easily duplicated. If $\mathcal{U}$'s transactions are anonymous, $\mathcal{U}$ might spend a coin more than once without any risk of being prosecuted. A solution to this problem would be to make payments on-line again while retaining anonymity. This way, $\mathcal{B}$ and $\mathcal{S}$ could immediately deny acceptance of a coin that has already been spent. $\mathcal{U}$ would still be anonymous, but not able to double-spend.

There are other approaches to the problem of double-spending, which permit the system to be kept off-line:

- Double-spender identification after the fact can be provided by mechanisms which protect the anonymity of honest users, but result in "automatic" deanonymization of double-spenders.

  One-show blind signatures use a cut-and-choose-like [CFN88] approach: At withdrawal, $\mathcal{U}$ blinds k values each of which embeds his identity; $\mathcal{B}$ randomly chooses $k/2$ of the values, and $\mathcal{U}$ has to unblind them to prove their correct construction. Like this, the probability that $\mathcal{U}$ has embedded his identity correctly in the remaining $k/2$ values can be made arbitrarily large by choosing k appropriately. For a reasonable security, the amount of computation needed is quite considerable. At payment, $\mathcal{U}$ gives to $\mathcal{S}$ some data that for itself doesn't compromise $\mathcal{U}$'s anonymity; double-spending will result in two pieces of such data, which will reveal $I_{\mathcal{U}}$.

  Given two different payment transcripts, *restrictive blind signatures* also provide deanonymization. They are more efficient than one-show blind signatures and will be presented in Section 3.6.

- *tamper-resistant wallets* offer prior restraint of double-spending by keeping track of which coins have already been spent and forbidding their further use (actually, one wouldn't need "coins" anymore; it would suffice to update a balance variable in the wallet). Tamper-resistance isn't easy to implement (especially in small devices, and wallets should be small), thus double-spender identification after the fact by means as above should always be added.

3.6 Restrictive Blind Signatures

The term "restrictive blind signature" was first introduced in [Bra93] together with an e-cash system providing double-spender identification without the use of cut-and-choose techniques. Another scheme of this type was presented in [Fer93]. The key concept is to restrict the blinding process, i.e. ensure that the blinded message still contains the original information. Like this, the signer regains some control over what he signs. In other words, the restrictiveness property means that the signer is assured that the verifier performs the blinding in the prescribed manner.

The application to electronic payment systems is as follows: At withdrawal, $\mathcal{U}$ includes his identity $I_{\mathcal{U}}$ in the information that, together with the blindly issued signature by $\mathcal{B}$, will form the coin. $\mathcal{B}$ is unable to see the final coin and thus can't verify that $I_{\mathcal{U}}$ is contained in it; the restrictive signature scheme guarantees this.

Additionally, during payment, $\mathcal{U}$ has to respond to a random challenge c sent by $\mathcal{S}$. The response provides additional information about the internal structure of the coin. However, *one* such piece of information doesn't compromise $\mathcal{U}$'s anonymity. Should $\mathcal{U}$ ever double-spend a coin, with high probability the challenges used during payment and thus also the responses will be different. Two different responses for the same coin then allow the identity of $\mathcal{U}$ to be extracted from the coin.

With restrictive blind signatures, double-spending is traceable, but only after it has occurred.

3.6.1 The System from [Bra93]

In the following, Brands' anonymous off-line system from [Bra93] is presented. It is quite efficient and has been object of thorough investigation; it hasn't been broken, but its security is not provable. More precisely, coins are unforgeable in the random oracle model, but the restrictiveness, i.e. the infeasibility of obtaining untraceable coins, can't be proven. Some of the systems presented in the following chapter are based on Brands' scheme.

System Setup

$\mathcal{B}$ chooses a group G_q of prime order q such that computing discrete logarithms in G_q is infeasible. In [Bra93], Brands chose for explicitness the unique subgroup G_q of prime order q of the multiplicative group $\mathbf{Z}_p^*$, where p is a prime such that $q|(p-1)$.

$\mathcal{B}$ generates at random generators g, g_1, g_2 of G_q, collision-resistant one-way hash functions $\mathcal{H}, \mathcal{H}_0$, and a secret key $x \in_R Z_q^*$. The corresponding public key is $h := g^x$ and is published together with g, g_1, g_2 and $\mathcal{H}, \mathcal{H}_0$. It is assumed that computing discrete logarithms of the generators with respect to each other is infeasible for all parties.

Account Opening

At account opening, $\mathcal{U}$ chooses a secret value $u_1 \in \mathbf{Z}_q$ with $g_1^{u_1} g_2 \neq 1$ [2]; $\mathcal{U}$ computes his public identity or "account number" as $I := g_1^{u_1}$ and transmits it to $\mathcal{B}$; additionally, $\mathcal{U}$ proves knowledge of u_1.

All coins withdrawn by $\mathcal{U}$ will consist of a blind Chaum-Pedersen signature on the message (Ig_2), and thus contain $\mathcal{U}$'s identity. The restrictiveness added to the signature scheme guarantees that I will "survive" the blinding process. This is achieved by requiring $\mathcal{U}$ to know (at payment time) a representation

$$m' = g_1^{y_1} g_2^{y_2}$$

of the blinded message $m' = (Ig_2)^s g^t = g_1^{u_1 s} g_2^s g^t$. Hence, $\mathcal{U}$ has to choose $t := 0$ (recall that the blind Chaum-Pedersen signature scheme works also for this choice). For details see [Pfi99].

$\mathcal{B}$ computes the (repeatedly used) signature commitment $z := (Ig_2)^x$ and sends it to $\mathcal{U}$. Alternatively, $\mathcal{U}$ computes it by himself as $z := h_1^{u_1} h_2$, where $h_1 := g_1^x$, $h_2 := g_2^x$ are additional public keys of $\mathcal{B}$.

Withdrawal

In the withdrawal protocol in Figure 3.3, $\mathcal{B}$ generates $w \in_R \mathbf{Z}_q$ and sends the values a, b. $\mathcal{U}$ chooses a random blinding factor s which he keeps secret and computes $A := (Ig_2)^s$; this is the blinded message that embeds $\mathcal{U}$'s identity. $\mathcal{U}$ also chooses random values x_1, x_2 which are used to compute B. These values are an addition to the Chaum-Pedersen scheme; (x_1, x_2) can be seen as a secret coin key that is needed for payment, B is the corresponding public value. B is included in the hash value c' as part of the "extended" blinded message $A||B$.

If both participants follow the withdrawal protocol, $\mathcal{U}$ will end up with a valid signature $sig(A, B) = (z', a', b', r')$; the verification equations

$$g^{r'} \stackrel{?}{=} h^{\mathcal{H}(A||B||z'||a'||b')} a' \text{ and } A^{r'} \stackrel{?}{=} z'^{\mathcal{H}(A||B||z'||a'||b')} b'$$

hold:

$$\begin{aligned} g^{r'} &= g^{ru+v} \\ &= g^{cxu+wu+v} \\ &= g^{c'x} a^u g^v \\ &= h^{\mathcal{H}(A||B||z'||a'||b')} a' \end{aligned}$$

$$\begin{aligned} A^{r'} &= A^{ru+v} \\ &= A^{cxu+wu+v} \\ &= A^{c'x} b^{su} A^v \\ &= z'^{\mathcal{H}(A||B||z'||a'||b')} b'. \end{aligned}$$

[2] Otherwise, $\mathcal{U}$ would know two representations of $g_1^{u_1} g_2$ $((u_1; 1)$ and $(0; 0))$ and could double-spend without being identified.

$$\begin{array}{ccc}
\mathcal{U} & & \mathcal{B} \\
(z := (Ig_2)^x) & & (z := (Ig_2)^x) \\
\hline
 & & w \in_R \mathbf{Z}_q \\
 & & a := g^w, b := (Ig_2)^w \\
 & \xleftarrow{\quad a, b \quad} & \\
s \in_R \mathbf{Z}_q^* & & \\
A := (Ig_2)^s & & \\
z' = z^s & & \\
x_1, x_2, u, v \in_R \mathbf{Z}_q & & \\
B := g_1^{x_1} g_2^{x_2} & & \\
a' := a^u g^v & & \\
b' := b^{su} A^v & & \\
c' := \mathcal{H}(A||B||z'||a'||b') & & \\
c := c'/u & & \\
 & \xrightarrow{\quad c \quad} & \\
 & & r := cx + w \\
 & \xleftarrow{\quad r \quad} & \\
g^r \stackrel{?}{=} h^c a,\ (Ig_2)^r \stackrel{?}{=} (z)^c b & & \\
r' := ru + v & & \\
\hline
(sig(A, B) = (z', a', b', r')) & &
\end{array}$$

Figure 3.3: Withdrawal in the system from [Bra93].

Payment

In the payment protocol in Figure 3.4, $\mathcal{U}$ provides $\mathcal{S}$ with $A, B, sig(A, B)$. $\mathcal{S}$ verifies $A \overset{?}{\neq} 1$ [3] and computes a challenge d based on the coin and on information specific to this payment (his identity $I_{\mathcal{S}}$, date/time). $\mathcal{U}$ responds with r_1, r_2, and $\mathcal{S}$ verifies $sig(A, B)$ and the correctness of $\mathcal{U}$'s response.

Double-spender Identification

If a coin is deposited more than once, $\mathcal{B}$ obtains at least two payment transcripts for the same coin. If $I_{\mathcal{S}}$, the date/time, and the response of $\mathcal{U}$ are equal in both transcripts, $\mathcal{S}$ is trying to deposit a coin twice. In the case that the coin was actually double-spent by $\mathcal{U}$, the challenges from payment will be different (otherwise $\mathcal{U}$ would have found a collision for the hash function $\mathcal{H}$), and therefore with high probability also the responses (r_1, r_2) and (r'_1, r'_2). $\mathcal{B}$ can then compute

$$\frac{r_1 - r'_1}{r_2 - r'_2} = u_1,$$

thus obtaining the account number $I = g_1^{u_1}$ of the double-spender and u_1, which serves as a proof of double-spending.

[3] Otherwise, $\mathcal{U}$ might have chosen $\mathsf{s} = 0$ during withdrawal, thus being able to prevent double-spender identification by responding $\mathsf{r}_1 := 0 + \mathsf{x}_1$, $\mathsf{r}_2 := 0 + \mathsf{x}_2$ in the payment protocol.

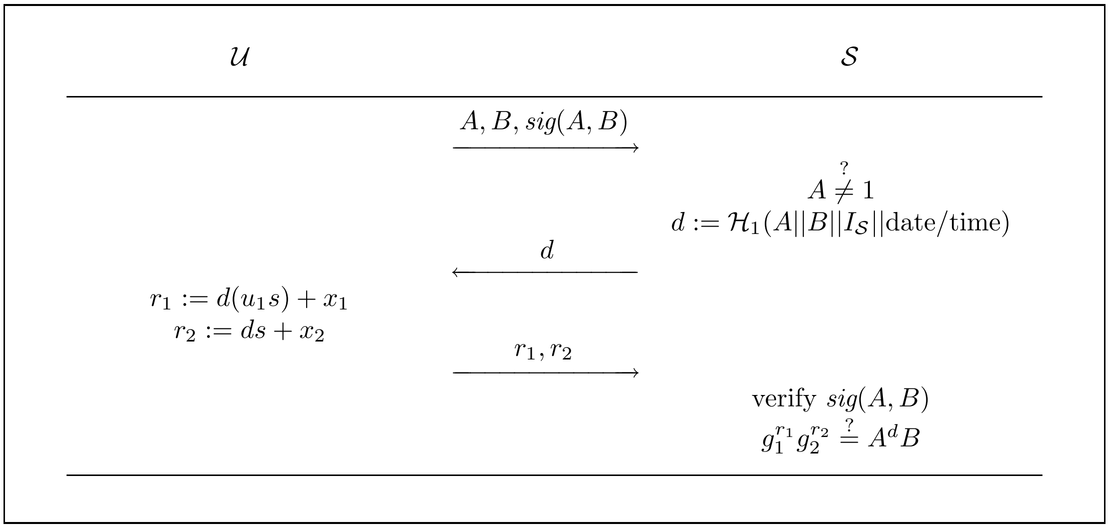


Figure 3.4: Payment in the system from [Bra93].

3.7 Misuse of Anonymity

In the following, the possibilities to misuse the anonymity of e-cash systems are listed, sorted by the misusing party. Ways to prevent the misuse are marked by $\star$. As we'll see, implementing anonymity revocation is sometimes the only solution, which leads us to the next chapter.

Fraudulent $\mathcal{U}$

- Overspending: $\mathcal{U}$ spends coins for an amount exceeding their value.

  $\star$ Implement adequate methods, e.g. as described in Section 3.5.

Fraudulent $\mathcal{S}$

- Money laundering: $\mathcal{S}$ obtains coins from an illegal activity and issues a fictitious bill to conceal their origin.

  $\star$ Implement anonymity revocation.

Fraudulent Outsider

A criminal $\mathcal{C}$ not being any of the above parties, but possibly having a bank account, might try the following attacks:

- Theft of coins from $\mathcal{U}$.

  $\star$ Make coins spendable only by $\mathcal{U}$; this does not necessarily contradict anonymity, since $\mathcal{U}$ may prove coin ownership at payment via a blind proof.

- Bank blackmailing: see "perfect blackmailing" below.

- User blackmailing: $\mathcal{C}$ forces $\mathcal{U}$ to withdraw coins and to hand them to him in a way that he will be able to spend them. The crucial point is that the handing over of the digital coins might not require any physical contact; then it can be accomplished by, e.g., a newspaper. Thus, the part of a blackmailing operation that poses the major threat of being caught to $\mathcal{C}$ is rendered totally safe for him. In addition he can spend the coins without any fear as long as he doesn't double-spend; if

$\mathcal{B}$ had any means of tracing the coins, he could do so also with regular coins and thus trace honest users.

$\star$ Implement anonymity revocation.

3.7.1 Perfect Blackmailing

Perfect blackmailing as introduced in [vSN92] actually means extorting coins from $\mathcal{B}$. However, one could also imagine that the blackmailed entity is $\mathcal{U}$, and $\mathcal{B}$ cooperates at $\mathcal{U}$'s request (and at $\mathcal{U}$'s promise to reimburse $\mathcal{B}$). The blackmailing attack presented in [vSN92] is shortly shown here:

Regular Withdrawal of a Coin

Let f be a one-way function and let n be an RSA modulus, i.e. $n = pq$ for two primes p, q where n is public and p, q are known only to $\mathcal{B}$. $\mathcal{B}$ issues coins as follows:

1. $\mathcal{U}$ randomly chooses a value x and a blinding factor r, computes $f(x)$, blinds it to $B := r^3 f(x) \pmod{n}$, and sends B to $\mathcal{B}$.
2. $\mathcal{B}$ computes $D := B^{1/3} \pmod{n}$, debits $\mathcal{U}$'s account, and sends D to $\mathcal{U}$.
3. $\mathcal{U}$ unblinds D to $C := D/r \pmod{n} = \big(f(x)\big)^{1/3} \pmod{n}$.

The pair (x, C) now represents a digital coin. $\mathcal{B}$ hasn't seen x or C and therefore can't link the coin to the withdrawal. The validity of the coin is verified by computing

$$f(x) \overset{?}{=} C^3.$$

NB. Computing D requires the ability to solve the RSA problem (Subsection 2.3.5), which is feasible if $\phi(n) = (p-1)(q-1)$ is known, i.e. if one knows the factorization of n.

The Attack

The criminal $\mathcal{C}$ proceeds as follows to obtain p coins:

1. Randomly choose $\{x_1, \ldots, x_p\}, \{r_1, \ldots, r_p\}$.
2. Compute $\{B_j | B_j = r_j^3 f(x_j) \pmod{n},\, j \in \{1, \ldots, p\}\}$ and send this set to $\mathcal{B}$.
3. Force $\mathcal{B}$ to compute $\{D_j | D_j = B_j^{1/3} \pmod{n},\, j \in \{1, \ldots, p\}\}$ and to publish this set in a newspaper.
4. Buy the newspaper and compute $\{C_j | C_j = D_j / r_j \pmod{n},\, j \in \{1, \ldots, p\}\}$.

Now (x_j, C_j) is a valid digital coin for $j \in \{1, \ldots, p\}$.

Blackmailing of $\mathcal{B}$ was presented in a strongly simplified anonymous system without anonymity revocation. Nevertheless, when looking, in the following chapters, at ways to augment the security of e-cash systems, it should be clear that no matter how sophisticated the system, an attacker who succeeds in forcing $\mathcal{B}$ to do as he wishes will always achieve his goal. As with conventional cash, $\mathcal{B}$ must resist all threats.

Chapter 4

Escrow-based E-Cash Systems

In the previous chapter, anonymous off-line e-cash was presented. User anonymity is a desirable feature; however, in certain situations it can be equally desirable to be able to revoke this anonymity. Such circumstances are

- user blackmailing
- money laundering
- otherwise suspect withdrawals or payments, e.g. by users suspected of criminal activities.

Two different kinds of anonymity revocation are useful, namely

- coin (or withdrawal-based) tracing: the ability to recognize coins that resulted from a suspect withdrawal (prevents misuse of anonymity for blackmailing)
- owner (or payment-based) tracing: the ability to identify users that carried out suspect payments (prevents misuse of anonymity for money laundering).

So-called *fair* blind signatures, introduced in [CPS95], make both types of anonymity revocation possible. They allow the signer ($\mathcal{B}$) to link unblind message-signature pairs (coins) to the corresponding blind ones (owner tracing) or to extract the unblind message-signature pair from the blind one (coin tracing). For this the signer needs some information, typically given to him by a trusted third party $\mathcal{T}$, called trustee.

E-cash systems that use fair blind signatures are (not surprisingly) called fair e-cash systems. Another common (and more descriptive) name is escrow-based e-cash [1]; the word "escrow" suggests that the information needed for tracing is deposited somewhere and accessed when necessary.

The trustee $\mathcal{T}$ should have no other role but the one of helping in tracing; like this, a clash of interests is prevented. Different escrow-based systems with trustee ([CMS96, FTY96, FTY98, dST98]) are presented and their security is investigated in this chapter.

Instead of having to trust a single entity, one can distribute the revocation capability among several trustees, allowing only the collusion of a certain number of them to deanonymize. In addition to limiting trust, this can also augment the availability (not all trustees are required for deanonymization). Dividing trust among many parties is not further investigated here; interested readers see [Pfi98, Cam98].

If having to trust any third entity is unwanted, self-escrowing [PS00] offers a solution. It provides coin tracing, thus protecting against user blackmailing; it doesn't require a trusted third party: the user himself has the secret information needed for tracing his coins. Self-escrowing is addressed in Chapter 5.

[1] The shorter term "escrowed e-cash" is misleading, as the coins themselves aren't escrowed anywhere.

4.1 Security Requirements for Escrow-based E-Cash

With the addition of anonymity revocation to an e-cash system, some new security requirements arise and old requirements gain new aspects. In the following, the major changes are presented.

Requirements of $\mathcal{T}$

$\mathcal{T}$'s only duty and therefore only interest is that of revoking anonymity. Hence, $\mathcal{T}$ wants to be always able to do so. Therefore,

- it should be infeasible for anyone to obtain untraceable coins,
- in systems where $\mathcal{T}$ actively participates in the withdrawal of coins, so-called blindfolding of $\mathcal{T}$ should be infeasible. This term means forcing someone to participate in a modified protocol, here in one that results in untraceable coins.

Requirements of $\mathcal{U}$

An honest $\mathcal{U}$ doesn't want to lose his anonymity; presuming $\mathcal{U}$ doesn't double-spend, this is possible only if $\mathcal{T}$ revokes it. There are two possible scenarios for this:

- $\mathcal{B}$ falsely convinces $\mathcal{T}$ that $\mathcal{U}$'s anonymity must be revoked. This can be prevented by employing methods that provide automatic double-spender identification. Then, $\mathcal{B}$ isn't allowed to ask $\mathcal{T}$ for anonymity revocation.
- Some authority who is allowed to demand anonymity revocation falsely asks $\mathcal{T}$ to revoke $\mathcal{U}$'s anonymity. The authority can only be trusted by $\mathcal{U}$ not to do this; therefore only trusted public authorities (e.g. judges) should have the power to demand anonymity revocation.
- $\mathcal{T}$ falsely identifies an honest $\mathcal{U}$. Although the probability for this event should be small (as $\mathcal{T}$ is called "trustee" and should be trusted also by $\mathcal{U}$), it can be prevented by making the anonymity revocation protocol such that the correctness of its result can be proven to any third party (e.g. a judge).

Requirements of $\mathcal{B}$

- $\mathcal{B}$ doesn't want to have to trust $\mathcal{T}$ for anything else but tracing; specifically, $\mathcal{T}$ shouldn't be able to issue coins.
- Blindfolding $\mathcal{B}$ to obtain untraceable and thus double-spendable coins should be infeasible.

4.2 On-line vs. Off-line Trustee

The trustee $\mathcal{T}$ will revoke $\mathcal{U}$'s anonymity under special circumstances. To be able to do so, $\mathcal{T}$ has to be able to link coins to withdrawals.

At coin withdrawal, $\mathcal{T}$ may be

- on-line: $\mathcal{T}$ performs the blinding and can therefore trivially trace, as he has seen the coin in its unblinded form,
- off-line: $\mathcal{U}$ encrypts the tracing information with $\mathcal{T}$'s public key, gives the encrypted information to $\mathcal{B}$ and proves that $\mathcal{T}$ will be able to trace (i.e. that the public key is indeed $\mathcal{T}$'s and that the information encrypted is correct and somehow linked to the coin, e.g. embedded in the coin).

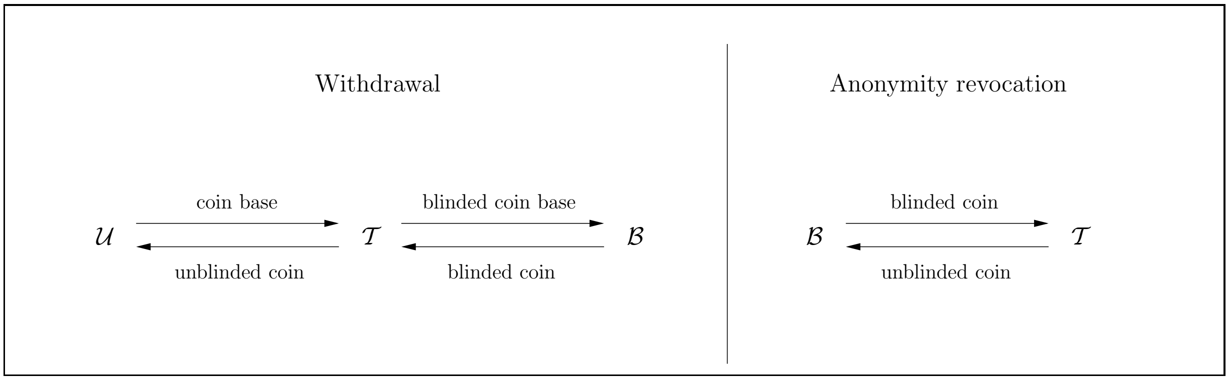


Figure 4.1: Trustee on-line at withdrawal.

On- and off-line trustee schemes are sketched in Figures 4.1 and 4.2. (Recall that the "coin base" is, e.g., a random number; $\mathcal{U}$ then ends up with the coin, consisting of this number and a signature by $\mathcal{B}$ on it.) The anonymity revocation presented there is coin tracing, i.e. $\mathcal{B}$ wants to know what the unblinded coin looks like. The other type of anonymity revocation, owner tracing, would be performed in the on-line case by $\mathcal{B}$ handing to $\mathcal{T}$ the unblinded coin used in a payment and receiving the corresponding blinded coin from the withdrawal; with $\mathcal{T}$ off-line, $\mathcal{B}$ would give $\mathcal{T}$ some part of the payment transcript obtained from $\mathcal{S}$. By use of this information and of his secret key, $\mathcal{T}$ could then establish the link to the withdrawal.

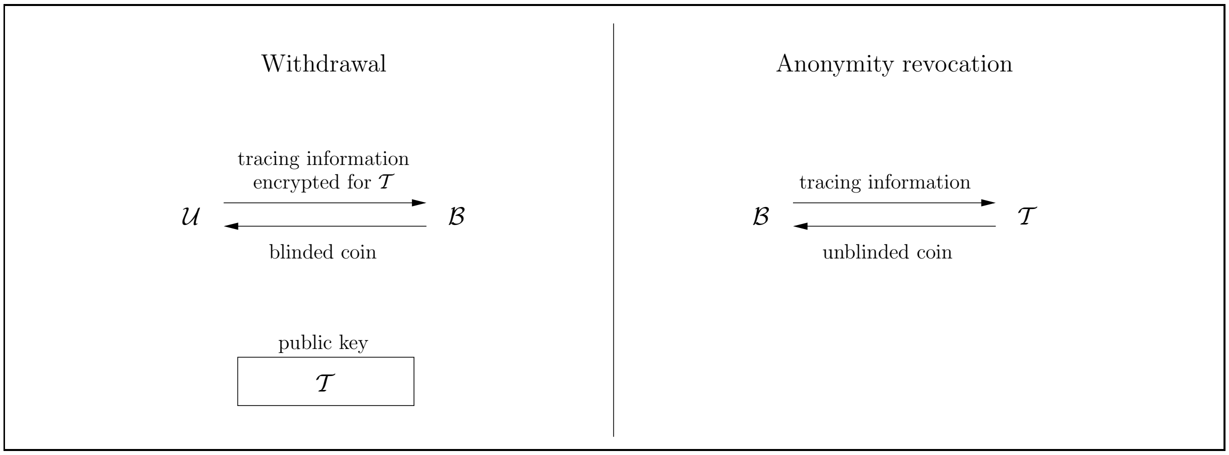


Figure 4.2: Trustee off-line at withdrawal.

Clearly, with $\mathcal{T}$ being off-line, the amount of computation needed during withdrawal rises. Not only has the encryption to be computed, but also its correct construction has to be proven in a separate protocol. However, systems with an off-line trustee have been more intensely studied as it is desirable that $\mathcal{T}$ needs to participate in transactions only for anonymity revocation. The greater size of the communication infrastructure for schemes with on-line trustees can easily be imagined.

Another variant of escrow-based e-cash is that of $\mathcal{T}$ certifying (i.e. digitally signing) pseudonyms to be used by $\mathcal{U}$. This approach is sketched in Figure 4.3: $\mathcal{B}$ receives only the pseudonym at withdrawal; if tracing is required, $\mathcal{T}$ is able to link the pseudonym to the real identity of $\mathcal{U}$. In such a system, payments made with coins that were withdrawn under the same pseudonym are linkable, which might be undesirable. On the other hand, the radical approach of choosing a different pseudonym for each coin to achieve unlinkability needs as much involvement of $\mathcal{T}$ as on-line escrowing. The difference is that several pseudonyms may be certified at once and saved by $\mathcal{U}$ for later use, cutting down the number of times $\mathcal{U}$ and $\mathcal{T}$ have to interact.

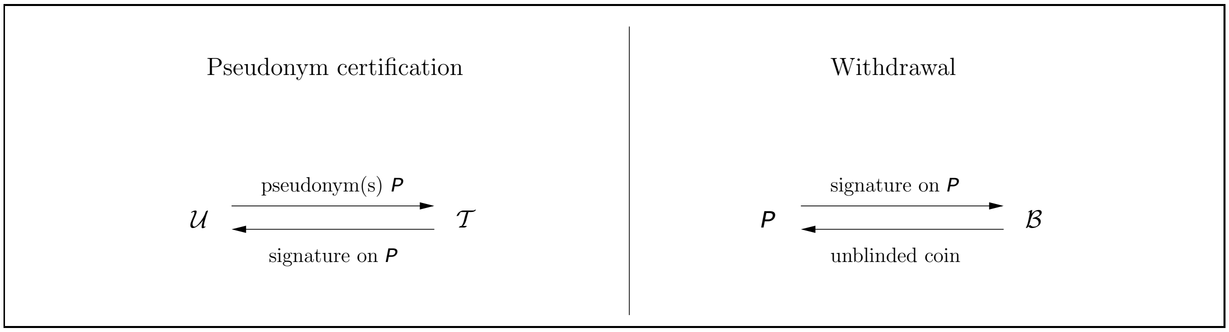


Figure 4.3: Trustee certifying pseudonyms.

4.3 Systems with Passive Trustee

Apart from the “on-line/off-line at withdrawal” classification above, $\mathcal{T}$ may be needed in the following phases:

- at account opening or at regular intervals (e.g. to certify pseudonyms),
- at payment time; this is imaginable only in very specific systems, as $\mathcal{T}$ has to be highly available,
- only for anonymity-revocation; in this case $\mathcal{T}$ is called a *passive* trustee.

In the following, different systems with passive trustee will be analyzed:

- The system of Camenisch, Maurer, and Stadler presented in [CMS96]
- The system of Frankel, Tsiounis, and Yung presented in [FTY96].
- The system of Frankel, Tsiounis, and Yung presented in [FTY98].
- The system of de Solages and Traoré presented in [dST98].

In [CMS96], both an on-line and an off-line system were presented, whereas [FTY96], [FTY98], and [dST98] deal only with off-line systems (on-/off-line here referring to the role of $\mathcal{B}$ during payment).

No Unconditional Unlinkability

In [FTY96] it was proven that unconditional unlinkability is impossible when the trustee is passive. The key idea is that unconditional unlinkability would imply that the encryption of the tracing information is unconditional, e.g. a one-time pad. As is known from information theory, $\mathcal{T}$ and $\mathcal{U}$ would then have to share information proportional to the size of all the plaintexts, i.e. the coins. This contradicts the idea that $\mathcal{U}$ needs only $\mathcal{T}$'s public key (which is of fixed length and ideally the same for all users). Thus, the unlinkability in passive trustee systems can only be computational.

4.3.1 The On-line System from [CMS96]

The system is similar to the one from [Bra93]; the main components of a coin are values (h_p, z_p) such that $h_p = z_p^x$ (where x is $\mathcal{B}$'s secret key), a proof W of this fact, and a proof V (on the structure of the coin) which guarantees that anonymity revocation is possible. The proof W is issued by $\mathcal{B}$ during withdrawal, while V is computed by $\mathcal{U}$ and verified by $\mathcal{S}$ at payment time.

The trustee isn't trusted for anything else but tracing; double-spending is no issue in an on-line system.

System Setup

$\mathcal{B}$ chooses a finite group G of prime order q such that computing discrete logarithms in G is infeasible. As q is prime, any $a \in G$ is a generator of G; generators g, g_1, g_2 are chosen in a publicly verifiable pseudorandom manner to ensure that the discrete logarithms of them with respect to each other are unknown. $\mathcal{B}$ also chooses a secret key $x \in_R \mathbf{Z}_q^*$ and computes the corresponding public key as $y := g^x$. G, g, g_1, g_2, and y are published.

$\mathcal{T}$ chooses a secret key $\tau \in_R \mathbf{Z}_q^*$; the corresponding public key is $y_\mathcal{T} = g_2^\tau$, which is published.

Withdrawal

The withdrawal protocol is shown in Figure 4.4; $\mathcal{U}$ chooses a random coin number $c\#$ and a random exponent $\alpha \in_R \mathbf{Z}_q^*$. This value serves various purposes:

1. $\mathcal{U}$ uses it to compute the "coin base" $h_w := g_1^{1/\alpha} g_2$.
2. The unblinded version h_p of h_w will be $h_p := h_w^\alpha$. (The indices w, p stand for **w**ithdrawal and **p**ayment; thus, h_w is the blind value shown to $\mathcal{B}$ during withdrawal, and h_p is the unblind value used for payment.)
3. The tracing information is computed as $d := y_\mathcal{T}^\alpha$, which can be seen as a Diffie-Hellman-type encryption of h_p for $\mathcal{T}$.

After having computed h_w and d, $\mathcal{U}$ creates a proof of equality of logarithms

$$U := \mathit{Proof}_{EQLOG}(g_1, (h_w/g_2), d, y_\mathcal{T}) = \mathit{PLOGEQ}(\epsilon, g_1, (h_w/g_2), d, y_\mathcal{T}) = (c_U, s_U)$$

as described in Subsection 2.7.4. It serves to prove

$$\log_{g_1}(h_w/g_2) = \log_d y_\mathcal{T},$$

and thus the correct construction of h_w and d to $\mathcal{B}$. $\mathcal{B}$ verifies U by checking

$$c_U \overset{?}{=} \mathcal{H}(\epsilon||g_1||d||(h_w/g_2)||y_\mathcal{T}||g_1^{s_U}(h_w/g_2)^{c_U}||d^{s_U} y_\mathcal{T}^{c_U}),$$

and engages in an interactive blind signature protocol with $\mathcal{U}$, referred to in [CMS96] as "protocol P" [2], which results in the pair $(h_p, z_p) = (h_w^\alpha, z_w^\alpha)$ and a proof

$$W := \mathit{Proof}_{EQLOG}(g, y, h_p, z_p) = \mathit{PLOGEQ}(c\#, g, y, h_p, z_p) = (c, s)$$

which on the one hand is a blind Schnorr signature on the message $c\#$ (the coin number), and on the other hand proves that

$$\log_g y = \log_{h_p} z_p,$$

and thus that $z_p = h_p^x$. W is verified by checking

$$c \overset{?}{=} \mathcal{H}(c\#||g||h_p||y||z_p||g^s y^c||h_p^s z_p^c).$$

$\mathcal{B}$ stores d in his withdrawal database.

The actual coin consists of the coin number $c\#$, the values h_p, z_p, W, and a proof of knowledge of a discrete logarithm

$$V := \mathit{Proof}_{KLOG}(g_2, h_p/g_1) = \mathit{PKLOG}(\epsilon, g_2, h_p/g_1) = (c_V, s_V)$$

that $\mathcal{U}$ computes by himself before payment. V proves that h_p is of the form $h_p = g_1 g_2^\xi$ for some ξ known to $\mathcal{U}$, and thus prevents the attack of choosing $h_p := h_w^\alpha g^\beta$ and $z_p := z_w^\alpha y^\beta$ for some $\beta \neq 0$ during

[2] The verifications performed by $\mathcal{U}$ at the end of the protocol were omitted in [CMS96].

withdrawal. Such an attack would yield a valid untraceable coin [3]. We'll see why furthermore $\xi = \alpha$ holds in the subsection "Security for $\mathcal{T}$" below.

The verification equation for V is

$$c_V \overset{?}{=} \mathcal{H}(\epsilon || g_2 || h_p/g_1 || g_2^{s_V} (h_p/g_1)^{c_V}).$$

$\mathcal{U}$		$\mathcal{B}$
random coin number $c\#$		
$\alpha \in_R \mathbf{Z}_q^*$		
$h_w := g_1^{1/\alpha} g_2$		
$d := y_{\mathcal{T}}^{\alpha}$		
$U := Proof_{EQLOG}(g_1, (h_w/g_2), d, y_{\mathcal{T}})$		
	$\xrightarrow{h_w, d, U}$	
		verify U
	protocol P	
		$z_w := h_w^x$
		$\tilde{r} \in_R \mathbf{Z}_q$
		$\tilde{t}_g := g^{\tilde{r}}$
		$\tilde{t}_h := h_w^{\tilde{r}}$
	$\xleftarrow{z_w, \tilde{t}_g, \tilde{t}_h}$	
$\gamma, \delta \in_R \mathbf{Z}_q$		
$h_p := h_w^{\alpha}$		
$z_p := z_w^{\alpha}$		
$t_g := \tilde{t}_g g^{\gamma} y^{\delta}$		
$t_h := \tilde{t}_h^{\alpha} h_p^{\gamma} z_p^{\delta}$		
$c := \mathcal{H}(c\# \vert\vert g \vert\vert h_p \vert\vert y \vert\vert z_p \vert\vert t_g \vert\vert t_h)$		
$\tilde{c} := c - \delta$		
	$\xrightarrow{\tilde{c}}$	
		$\tilde{s} := \tilde{r} - \tilde{c}x$
	$\xleftarrow{\tilde{s}}$	
$g^{\tilde{s}} \overset{?}{=} \tilde{t}_g / y^{\tilde{c}}$ and $h_w^{\tilde{s}} \overset{?}{=} \tilde{t}_h / z_w^{\tilde{c}}$		
$s := \tilde{s} + \gamma$		
$W := (c, s)$		
$(c\#, h_p, z_p, \alpha, W)$		(d)

Figure 4.4: Withdrawal in the on-line system from [CMS96].

Payment

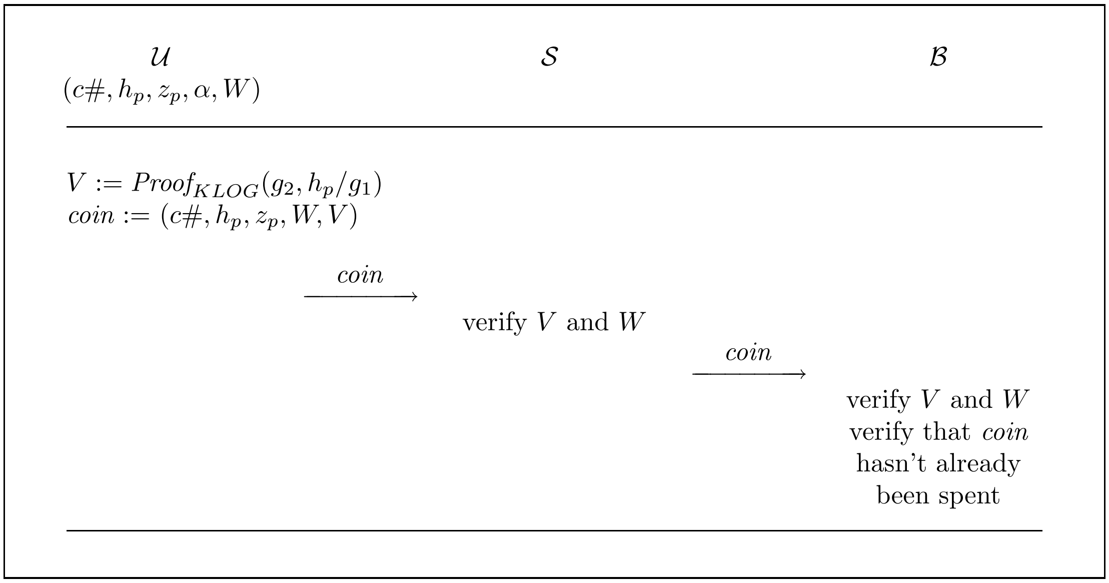
The payment protocol is shown in Figure 4.5; $\mathcal{U}$ sends the coin to $\mathcal{S}$, who verifies V and W by checking

$$c_V \overset{?}{=} \mathcal{H}\big(\epsilon || g_2 || (h_p/g_1) || g_2^{s_V} (h_p/g_1)^{c_V}\big)$$

[3] To check validity, execute the protocol with those values; all verification equations hold. To check untraceability, perform the computations for anonymity revocation described below: owner tracing would yield $(h_p/g_1)^{\tau} = (g_2^{\alpha} g^{\beta})^{\tau} \neq d := g_2^{\alpha}$; coin tracing would result in $g_1 d^{1/\tau} = g_1 g_2^{\alpha} \neq h_p := g_1 g_2^{\alpha} g^{\beta}$.

$$c \overset{?}{=} \mathcal{H}\big(c\#||g||y||h_p||z_p||g^s y^c||h_p^s z_p^c\big),$$

and passes the coin on to $\mathcal{B}$. In addition to verifying V and W, $\mathcal{B}$ also checks in his deposit database if the coin has already been spent.



$\mathcal{U}$ $(c\#, h_p, z_p, \alpha, W)$

$\mathcal{S}$

$\mathcal{B}$

$V := Proof_{KLOG}(g_2, h_p/g_1)$
$coin := (c\#, h_p, z_p, W, V)$

$\xrightarrow{coin}$

verify V and W

$\xrightarrow{coin}$

verify V and W
verify that *coin*
hasn't already
been spent

Figure 4.5: Payment in the on-line system from [CMS96].

Anonymity Revocation

The anonymity can be revoked by $\mathcal{T}$ by use of his secret key τ; the two different kinds of anonymity revocation are performed as follows:

- Payment-based (owner tracing): $\mathcal{T}$ is given an h_p observed in a payment; $\mathcal{T}$ computes $(h_p/g_1)^\tau = (g_2^\alpha)^\tau = d$, and $\mathcal{B}$ searches his withdrawal database for d.
- Withdrawal-based (coin tracing): $\mathcal{T}$ is given a d observed in a withdrawal; $\mathcal{T}$ computes $g_1 d^{1/\tau} = g_1 g_2^\alpha = h_p$, and h_p can be put on a blacklist for recognizing the coin when it is spent.

The equalities used above hold because for every triple (h_w, h_p, d) originating from a correct withdrawal it is true that

$$\alpha = (\log_{g_1}(h_w/g_2))^{-1} = \log_{y_\mathcal{T}} d = \log_{g_2}(h_p/g_1).$$

Security for $\mathcal{U}$

For payments to be unlinkable and anonymous, the protocol P has to be blind; this can be proven by showing that for every possible view of $\mathcal{B}$ and for every possible message-signature tuple there exists exactly one suitable tuple (α, γ, δ) of blinding factors. Given a view consisting of $h_w, z_w, \tilde{t}_g, \tilde{t}_h, \tilde{c}, \tilde{s}$ and a message-signature tuple $(c\#, h_p, z_p, c, s)$, the only possibility is

$$\begin{array}{rcl} \alpha & := & \log_{h_w} h_p = \log_{z_w} z_p \\ \gamma & := & s - \tilde{s} \\ \delta & := & c - \tilde{c} \end{array}$$

With these blinding factors, $\mathcal{U}$ would have computed

$$\begin{aligned} h_p &:= h_w^{\alpha} \\ z_p &:= z_w^{\alpha} \\ t_g &:= \tilde{t}_g g^{\gamma} y^{\delta} \\ t_h &:= \tilde{t}_h^{\alpha} h_p^{\gamma} z_p^{\delta} \\ c &:= \mathcal{H}(c\#||g||h_p||y||z_p||t_g||t_h) \end{aligned}$$

It remains to show that $t_g = g^s y^c$ and $t_h = h_p^s z_p^c$:

$$t_g = \tilde{t}_g g^{\gamma} y^{\delta} = g^{\tilde{r}+\gamma+x\delta} = g^{\tilde{r}+s-\tilde{s}+x(c-\tilde{c})} = g^{s+xc} g^{\tilde{r}-x\tilde{c}-\tilde{s}} = g^s y^c \text{ [4]}$$

$$t_h = \tilde{t}_h^{\alpha} h_p^{\gamma} z_p^{\delta} = h_p^{\tilde{r}+\gamma+x\delta} = h_p^{\tilde{r}+s-\tilde{s}+x(c-\tilde{c})} = h_p^{s+xc} h_p^{\tilde{r}-x\tilde{c}-\tilde{s}} = h_p^s z_p^c.$$

Thus, $c = \mathcal{H}(c\#||g||h_p||y||z_p||g^s y^c||h_p^s z_p^c)$ and the blinding factors (α, γ, δ) would in fact have resulted in the valid message-signature tuple $(c\#, h_p, z_p, c, s)$.

Although the blindness of protocol P is unconditional, $\mathcal{U}$'s anonymity is only computational, because the enrypted tracing information d could be used by $\mathcal{B}$ to link withdrawal and payment by testing

$$\log_{y_{\mathcal{T}}} d \stackrel{?}{=} \log_{g_2}(h_p/g_1).$$

This would be feasible if one could efficiently compute discrete logarithms. Being able to solve the Decision Diffie-Hellman problem would also be sufficient, the instance to solve would be

$$\text{given } (h_p/g_1) = g_2^{\alpha} \text{ and } y_{\mathcal{T}} = g_2^{\tau}, \text{ decide } d \stackrel{?}{=} g_2^{\tau\alpha}.$$

In [Cam98] it was stated that linking like this is even equivalent to the Decision Diffie-Hellman problem, but we can't see this.

The proof V from payment can't be linked to the corresponding withdrawal. Furthermore, in [CMS96] it was stated that $\mathcal{B}$'s view from protocol P and the value d obtained during withdrawal both for themselves don't compromise user anonymity, at least against a computationally limited $\mathcal{B}$. What happens if we allow simultaneous evaluation of both pieces of information and possibly also of the proof U? The value d is defined as the public key of the trustee taken to the power of $\mathcal{U}$'s secret value α. As long as we assume that solving the discrete logarithm or the Decision Diffie-Hellman problem is infeasible, d shouldn't compromise $\mathcal{U}$'s anonymity. The same holds for the proof U, which is assumed not to leak information about α under the mentioned assumptions.

As double-spending is impossible in an on-line system, false accuses of overspending and further overspending by other parties are no issue.

Security for $\mathcal{B}$

In the random oracle model, forging coins without any interaction with $\mathcal{B}$ is as hard as computing discrete logarithms [5]. However, it can't be proven that after ℓ withdrawals, an attacker can't obtain more than ℓ valid coins. It can only be stated that this seems infeasible.

Double-spending is no issue in an on-line system.

Security for $\mathcal{S}$

As the system is on-line, coins paid to $\mathcal{S}$ should be hard to steal or extort (deposit takes place immediately after or even during payment). There is no way to prevent false accusations of double-depositing. However, this is not a serious problem ($\mathcal{B}$ wouldn't usually make such false accusations, as there is no money to be gained) and can be solved easily (by including a signature by $\mathcal{S}$ in every deposit request).

[4] The last equality holds because $\tilde{s} := \tilde{r} - \tilde{c}x$.

[5] Security proofs in the random oracle model are addressed in detail in Chapter 6.

Security for $\mathcal{T}$

It should be infeasible to obtain untraceable coins. A coin becomes untraceable if $\mathcal{U}$ uses, instead of α, another blinding factor α' for the computation of h_p and z_p. The resulting coin would then be one with $h_p = g_1 g_2^{\alpha'}$ such that $\alpha' \neq \alpha$; thus, anonymity revocation with the value $d := y_{\mathcal{T}}^{\alpha}$ wouldn't yield h_p. In the withdrawal protocol, $\mathcal{U}$ obtains the values $\tilde{t}_g, \tilde{t}_h$; to circumvent revocation, he would have to compute t_h as

$$t_h := h_p^{\tilde{r}} h_p^{\gamma} z_p^{\delta},$$

thus replacing $\tilde{t}_h^{\alpha} = h_w^{\tilde{r}\alpha}$ by $h_p^{\tilde{r}}$.

In Subsection 6.4.6 of [Cam98], it was claimed that an algorithm that computes $h_p^{\tilde{r}}$ would solve the Diffie-Hellman problem. At first sight we didn't agree, since [Cam98] required that the algorithm outputs $g_1^{\tilde{r}}$ instead of $h_p^{\tilde{r}}$, and this seemed a stronger assumption. But assuming that the algorithm indeed outputs $h_p^{\tilde{r}} = (g_1 g_2^{\alpha'})^{\tilde{r}}$ for any $\alpha' \neq \alpha$ that $\mathcal{U}$ chooses, the value $g_1^{\tilde{r}}$ can be obtained by letting the algorithm run with $\alpha' = 0$, and the further argumentation from [Cam98] holds. Thus, it is as hard as the Diffie-Hellman problem to obtain untraceable coins.

4.3.2 The Off-line System from [CMS96]

In the off-line version of the system from [CMS96], the withdrawal and payment protocols are slightly modified, allowing $\mathcal{B}$ to identify double-spenders without the help of $\mathcal{T}$. The anonymity of honest users is maintained.

The proof V is redefined; to understand how the identification of double-spenders works, one has to keep in mind that in the on-line system, V is a Schnorr signature as presented in Subsection 2.5.4. During its generation, the signer $\mathcal{U}$ chooses a random value r to compute c as $c := \mathcal{H}(m||g^r)$. If two different messages are signed using the same value r, the secret key can be computed from the two signatures. Therefore it suffices to force $\mathcal{U}$ to compute V using the same r for a given coin. Double-spending will then result in $\mathcal{B}$ being able to compute the secret key used to create V, which is α; then $\mathcal{B}$ can look up $d = y_{\mathcal{T}}^{\alpha}$ in the withdrawal database, and thus identify $\mathcal{U}$.

In the off-line system, $\mathcal{U}$ has to choose r_p (which plays the role of r) and include $t_p := g_2^{r_p}$ instead of the coin number $c\#$ in the proof W during withdrawal. Since this is the only modification to the withdrawal protocol, the whole new protocol is not shown here.

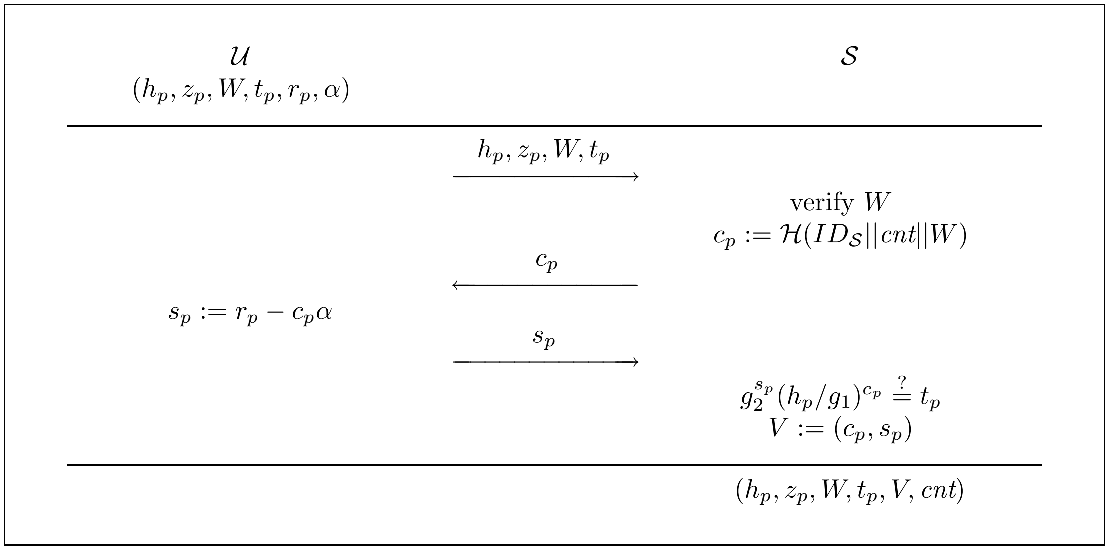

$\mathcal{U}$
$(h_p, z_p, W, t_p, r_p, \alpha)$
$\mathcal{S}$
h_p, z_p, W, t_p
verify W
$c_p := \mathcal{H}(ID_{\mathcal{S}}||cnt||W)$
c_p
$s_p := r_p - c_p\alpha$
s_p
$g_2^{s_p}(h_p/g_1)^{c_p} \overset{?}{=} t_p$
$V := (c_p, s_p)$
$(h_p, z_p, W, t_p, V, cnt)$

Figure 4.6: Payment in the off-line system from [CMS96].

The new payment protocol, however, is presented in Figure 4.6; $\mathcal{U}$ sends (h_p, z_p, W, t_p) to $\mathcal{S}$. After verifying

W, $\mathcal{S}$ provides $\mathcal{U}$ with the challenge $c_p := \mathcal{H}(ID_{\mathcal{S}}||cnt||W)$ and obtains the response s_p. The verification subsequently performed by $\mathcal{S}$ succeeds if s_p was computed correctly, because then

$$g_2^{s_p}(h_p/g_1)^{c_p} = g_2^{r_p - c_p\alpha}(h_w^\alpha/g_1)^{c_p} = g_2^{r_p - c_p\alpha} g_2^{c_p\alpha} = g_2^{r_p} = t_p.$$

The equalities $h_p = h_w^\alpha = g_1 g_2^\alpha$ hold because W proves that the withdrawal protocol was executed correctly.

The "new" proof V obtained by $\mathcal{S}$ is $V := (c_p, s_p)$; $\mathcal{S}$' final result $(h_p, z_p, W, t_p, V, cnt)$ can be passed to $\mathcal{B}$ for deposit who will be able to verify both W and V. The counter value *cnt* serves to protect $\mathcal{U}$ against $\mathcal{S}$ framing him as a double-spender; if *cnt* wasn't included, $\mathcal{S}$ could double-deposit the coin and say $\mathcal{U}$ paid twice with it. With *cnt*, $\mathcal{B}$ will only believe this if the values *cnt*, *cnt'* in the two payment transcripts differ, i.e. if $\mathcal{U}$ actually responded to two different challenges.

It should be noticed that in the off-line payment protocol, V is no longer a Schnorr signature. It may easily be transformed in one, however, by simply computing c_p as $c_p := \mathcal{H}(ID_{\mathcal{S}}||cnt||W||t_p)$. The authors of [CMS96] may have preferred their version for efficiency reasons. As pointed out by them, including W already links V to a specific withdrawal and therefore to this specific coin.

Double-spender Identification

If $\mathcal{U}$ spends a coin twice, $\mathcal{B}$ will end up with two different challenge-response pairs (c_p, s_p), (c'_p, s'_p) related to the coin. $\mathcal{B}$ then computes

$$\begin{aligned} s_p - s'_p &= r_p - c_p\alpha - r_p + c'_p\alpha = \alpha(c'_p - c_p) \\ \Rightarrow \quad \alpha &= \frac{s_p - s'_p}{c'_p - c_p} \\ \text{and} \quad d &= y_{\mathcal{T}}^\alpha, \end{aligned}$$

and looks d up in the withdrawal database, thus identifying $\mathcal{U}$.

Changes in Security with Respect to the On-line Scheme

Avoiding double-spender identification requires the ability to forge the signature V; this is believed to be infeasible.

$\mathcal{U}$ could be falsely accused of overspending by $\mathcal{B}$ if $\mathcal{B}$ was able to present two different signatures V for one coin although $\mathcal{U}$ spent the coin only once. This would mean being able to forge V, which is believed to be infeasible. Should $\mathcal{U}$ indeed overspend, then $\mathcal{B}$ could compute α and collude with some $\mathcal{S}$ to further overspend the coin.

Theft or extortion of coins paid to $\mathcal{S}$ is prevented by including $Id_{\mathcal{S}}$ in the payment transcript and thus making the coin depositable only by $\mathcal{S}$.

For all other aspects, the off-line scheme is as secure as the on-line system.

4.3.3 The Off-line System from [FTY96]

The key concept presented in [FTY96] is that of *indirect discourse proofs*. These allow one to prove that a third party will have some future capability, and this without active participation of the third party in question. For escrow-based off-line e-cash systems with trustee this means that $\mathcal{U}$ proves to $\mathcal{B}$ or $\mathcal{S}$ or both that $\mathcal{T}$ will be able to trace.

The indirect discourse proofs are based on homomorphic properties of exponentiation, as will be shown below. Obviously they shouldn't leak information to $\mathcal{B}$ or $\mathcal{S}$ or both that by itself (without $\mathcal{T}$'s cooperation) would allow tracing.

The scheme is based on a modification of [Bra93]. In [FTY96], it was presented step by step, first introducing the basic system which doesn't provide any kind of tracing, then making the necessary additions to this system. Here, the "final result" with owner and coin tracing is presented; for owner tracing, the payment receivers are assumed to be trusted [6], as in Section 5.1 of [FTY96].

To make understanding the protocols easier, they are shown divided in different parts as in the original article. In difference to the article, the payment receiver is denoted by $\mathcal{S}$ instead of $\mathcal{R}$. For some reason, the authors of [FTY96] decided to use the prime symbol in a manner complementary to [Bra93]; e.g., the variable a becomes a' and vice versa. This should be kept in mind when comparing the scheme here to the one described in Subsection 3.6.1.

System Setup

$\mathcal{B}$ chooses primes p, q with $p = \gamma q + 1$ for a specified small integer γ and the unique subgroup G_q of order q of $\mathbf{Z}_p$ with generators g, g_1, g_2 such that computing discrete logarithms in G_q is infeasible.

$\mathcal{B}$ also chooses secret keys $X_{\mathcal{B}}, X'_{\mathcal{B}} \in_R \mathbf{Z}_q$; there are corresponding public keys

$$h := g^{X_{\mathcal{B}}}, h_1 := g_1^{X_{\mathcal{B}}}, h_2 := g_2^{X_{\mathcal{B}}} \quad \text{and} \quad h' := g^{X'_{\mathcal{B}}}, h'_1 := g_1^{X'_{\mathcal{B}}}, h'_2 := g_2^{X'_{\mathcal{B}}}.$$

Additionally, collision-resistant one-way hash functions $\mathcal{H}, \mathcal{H}_0, \ldots$ are defined.

$\mathcal{T}$ chooses a secret key $X_{\mathcal{T}} \in_R \mathbf{Z}_q$; there are corresponding public keys $f_1 := g_1^{X_{\mathcal{T}}}, f_2 := g_2^{X_{\mathcal{T}}}$.

Account Opening

At account opening, $\mathcal{U}$ chooses a secret $u_1 \in \mathbf{Z}_q$ with $g_1^{u_1} g_2 \neq 1$ [7]; $\mathcal{U}$ computes his public identity as $I := g_1^{u_1}$ and proves to $\mathcal{B}$ that he knows the discrete logarithm of I with respect to g_1. $\mathcal{U}$ also computes $z' := h_1^{u_1} h_2 = (I g_2)^{X_{\mathcal{B}}}$ for future use during withdrawals: each coin will consist of a signature on the message $I g_2$, and $z' = (I g_2)^{X_{\mathcal{B}}}$ is the signature commitment; the coins will differ in the blinding of $I g_2$ and z'. As in Brands' system, $\mathcal{B}$ could also compute this value for $\mathcal{U}$ as $z' := (I g_2)^{X_{\mathcal{B}}}$.

Withdrawal

In the first part of the withdrawal protocol, shown in Figure 4.7, $\mathcal{U}$ chooses a random number s which he keeps secret; s will be the blinding factor that is used to blind z'. $\mathcal{U}$ computes $A := (I g_2)^s = A_1 A_2$ with $A_1 := I^s = g_1^{u_1 s}, A_2 = g_2^s$; thus, $\mathcal{U}$'s identity is embedded in A [8]. $\mathcal{U}$ chooses two random values x_1, x_2 and computes B_1, B_2, and $B = [B_1, B_2]$. Notice that the "splitting up" of A and B is the only change to the withdrawal protocol from [Bra93], where $\mathcal{U}$ computed $B = g_1^{x_1} g_2^{x_2}$. We believe that B is split up to make its representation (x_1, x_2) unique, and thus make manipulation even less probable [9].

$\mathcal{B}$ issues a restrictive blind signature on (A, B). As in Brand's scheme, it is a modified Chaum-Pedersen signature of the form $sig(A, B) = (z, a, b, r)$; the verification equations are

$$g^r \stackrel{?}{=} a h^{\mathcal{H}(A||B||z||a||b)} \text{ and } A^r \stackrel{?}{=} b z^{\mathcal{H}(A||B||z||a||b)}.$$

The second and third part of the withdrawal protocol contain the additions needed for coin tracing (owner tracing will be implemented by additions merely to the payment protocol). Before presenting those parts in detail, we sketch the principle:

- The value A_2 from the first part of the withdrawal protocol is encrypted to enc_{CT} [10] with the public key of $\mathcal{T}$. A_2 is handed to $\mathcal{S}$ at payment time, and thus the coin can be identified later, if $\mathcal{T}$ decrypts enc_{CT} with his secret key $X_{\mathcal{T}}$,

[6] This means that they are trusted not to collude with some other user $\mathcal{U}'$ to frame $\mathcal{U}$; this is no great risk, since $\mathcal{U}$ can prove his innocence and the payment receivers will be held liable for the attack. However, $\mathcal{U}$'s payments lose their anonymity and $\mathcal{U}'$ escapes identification; all this can be prevented by further measures described in Section 5.2 of [FTY96].

[7] Otherwise, $\mathcal{U}$ would know two representations of $g_1^{u_1} g_2$ ((u_1; 1) and (0; 0)) and could double-spend without being identified.

[8] We'll soon see what the values A_1, A_2 are used for.

[9] In Brands' system, an attacker capable of finding two representations of B could double-spend without being detected; he would do so by using different representations to compute his response to $\mathcal{S}$'s challenge at payment time.

[10] CT stands for coin tracing; we'll also have a separate encryption enc_{OT} for owner tracing.

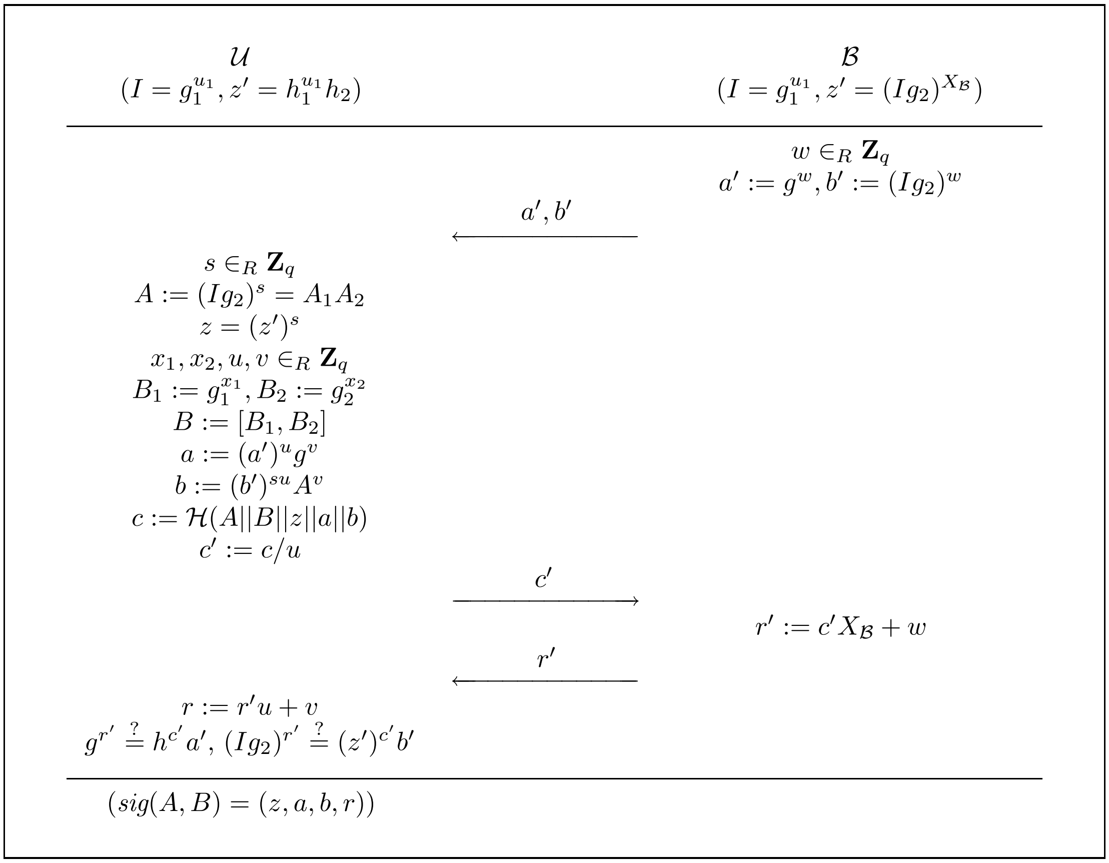

$\mathcal{U}$
$(I = g_1^{u_1}, z' = h_1^{u_1} h_2)$
$\mathcal{B}$
$(I = g_1^{u_1}, z' = (Ig_2)^{X_{\mathcal{B}}})$
$w \in_R \mathbf{Z}_q$
$a' := g^w, b' := (Ig_2)^w$
$\xleftarrow{a', b'}$
$s \in_R \mathbf{Z}_q$
$A := (Ig_2)^s = A_1 A_2$
$z = (z')^s$
$x_1, x_2, u, v \in_R \mathbf{Z}_q$
$B_1 := g_1^{x_1}, B_2 := g_2^{x_2}$
$B := [B_1, B_2]$
$a := (a')^u g^v$
$b := (b')^{su} A^v$
$c := \mathcal{H}(A||B||z||a||b)$
$c' := c/u$
$\xrightarrow{c'}$
$r' := c' X_{\mathcal{B}} + w$
$\xleftarrow{r'}$
$r := r'u + v$
$g^{r'} \overset{?}{=} h^{c'} a'$, $(Ig_2)^{r'} \overset{?}{=} (z')^{c'} b'$
$(sig(A, B) = (z, a, b, r))$

Figure 4.7: Withdrawal in the system from [FTY96], part 1.

- to prove the correct construction of enc_{CT}, an additional encryption A' of A_2 is provided,
- $\mathcal{B}$ verifies that both enc_{CT} and A' encrypt the same value and blindly signs A'; enc_{CT} is stored by $\mathcal{B}$ together with the withdrawal transcript,
- by unblinding A' and $\mathcal{B}$'s signature on it, $\mathcal{U}$ obtains a signature on $\bar{A}$ (the unblinded A'),
- at payment, $\mathcal{U}$ opens $\bar{A}$ and $\mathcal{S}$ verifies that $\bar{A}$ encrypts A_2.

By looking at the values that $\mathcal{B}$ and $\mathcal{S}$ respectively see, it is clear that $\mathcal{B}$ or $\mathcal{S}$ or both can't link withdrawals to payments. On the other hand, $\mathcal{B}$ verifies that enc_{CT} and A' encrypt the same value, $\mathcal{S}$ verifies that $\bar{A}$ encrypts A_2, and the restrictiveness of the blind signature scheme guarantees that $\bar{A}$ and A' encrypt the same value. Thus, enc_{CT} indeed encrypts A_2 and the coin can be traced with the help of $\mathcal{T}$.

Now, let's look at the second part of the withdrawal protocol, shown in Figure 4.8. In the course of it, $\mathcal{U}$ will supply $\mathcal{B}$ with the ElGamal encryption

$$enc_{CT} := (D_1', D_2') := (A_2 f_1^{m'}, g_1^{m'})$$

of A_2. Before computing enc_{CT}, $\mathcal{U}$ prepares additional values that will be used to convince $\mathcal{B}$ that enc_{CT} is indeed computed as above, of course without revealing its content. $\mathcal{U}$ chooses blinding factors n_1, n_2 and blinds A_2 to $A_1' := A_2^{n_1}$. With $A_2' := g_1^{n_1}$, the product $A' := A_1' A_2'$ is the second encryption of A_2 that $\mathcal{U}$ will open for $\mathcal{S}$ at payment time. $\mathcal{U}$ proves (using the Schnorr identification protocol as described in Subsection 2.7.3) that he knows how to represent A_1' with respect to g_2 and A_2' with respect to g_1; this ensures the correct construction of A'.

Next, $\mathcal{U}$ computes the encryption $enc_{CT} = (D_1', D_2')$ and engages in an indirect discourse proof with $\mathcal{B}$ to prove that

$\mathcal{U}$		$\mathcal{B}$
$n_1, n_2 \in_R \mathbf{Z}_q$, $N := n_1 n_2$		
$A'_1 := A_2^{n_1} = g_2^{sn_1}, A'_2 := g_1^{n_1}$		
$A' := A'_1 A'_2$		
$\mathcal{U}$ proves knowledge of representations of A'_1 and A'_2		
$m' \in_R \mathbf{Z}_q$		
$D'_1 := A_2 f_1^{m'}$, $D'_2 := g_1^{m'}$		
$enc_{CT} := (D'_1, D'_2)$		
	$\xrightarrow{A', A'_1, A'_2, enc_{CT}}$	
		$A' \overset{?}{=} A'_1 A'_2, D'_2 \overset{?}{\neq} 1$
		$s'_0, s'_1, s'_2 \in_R \mathbf{Z}_q$
		$\bar{D} := {D'_1}^{s'_0} g_1^{s'_1} {D'_2}^{s'_2}$
		$f'_1 := f_1^{s'_0} g_1^{s'_2}$
	$\xleftarrow{\bar{D}, f'_1}$	
$\bar{V} := \mathcal{H}_4\big((\bar{D})^{n_1}/(f'_1)^{n_1 m'}\big)$		
	$\xrightarrow{\bar{V}}$	
		$\bar{V} \overset{?}{=} \mathcal{H}_4\big((A'_1)^{s'_0}(A'_2)^{s'_1}\big)$
		(A'_1, A'_2, enc_{CT})

Figure 4.8: Withdrawal in the system from [FTY96], part 2.

$$
\begin{array}{ccc}
\mathcal{U} & & \mathcal{B} \\
(s, n_1, n_2, N = n_1 n_2, A') & & (A' = A'_1 A'_2) \\
\hline
 & & \omega \in_R \mathbf{Z}_q \\
 & & \alpha' := g^{\omega}, \beta' := A'^{\omega} \\
 & \xleftarrow{\quad \alpha', \beta' \quad} & \\
\upsilon, \nu \in_R \mathbf{Z}_q & & \\
\bar{A} := A'^{n_2} & & \\
\zeta' := h'^{s n_1}_2 h'^{n_1}_1 = A'^{X'_{\mathcal{B}}} & & \\
\zeta := \zeta'^{n_2} = A'^{X'_{\mathcal{B}} n_2} = \bar{A}^{X'_{\mathcal{B}}} & & \\
\alpha := \alpha'^{\upsilon} g^{\nu} & & \\
\beta := \beta'^{n_2 \upsilon} \bar{A}^{\nu} & & \\
\varsigma := \mathcal{H}(\bar{A} || N || \zeta || \alpha || \beta) & & \\
\varsigma' := \varsigma / \upsilon & & \\
 & \xrightarrow{\quad \varsigma' \quad} & \\
 & & \rho' := \varsigma' X'_{\mathcal{B}} + \omega \\
 & \xleftarrow{\quad \rho' \quad} & \\
\rho := \rho' \upsilon + \nu & & \\
g^{\rho'} \overset{?}{=} h'^{\varsigma'} \alpha', \; A'^{\rho'} \overset{?}{=} (\zeta')^{\varsigma'} \beta' & & \\
\hline
(sig(\bar{A}, N) = (\zeta, \alpha, \beta, \rho)) & &
\end{array}
$$

Figure 4.9: Withdrawal in the system from [FTY96], part 3.

- the encryption is based on $\mathcal{T}$'s public key,
- enc_{CT} and A'_1 both encrypt the same string ($\mathcal{B}$ doesn't see this string here and $\mathcal{U}$ could choose it arbitrarily for the moment; but at payment time, $\mathcal{S}$ will check that the string actually is A_2).

In the third and last part of the protocol, shown in Figure 4.9, $\mathcal{B}$ issues a restrictive blind signature on the values $(\bar{A}, N)$, where $\bar{A} := A'^{n_2}$, i.e. n_2 is used to blind A' to $\bar{A}$. This blinding's purpose is to guarantee unlinkability of withdrawal and payment; at payment and deposit, $\mathcal{S}$ and $\mathcal{B}$ will only see $\bar{A}$, which can't be linked to A'.

To make the signature on $(\bar{A}, N)$ differ from the one on (A, B), it is created with $\mathcal{B}$'s other secret key, $X'_{\mathcal{B}}$.

Payment

In the first part of the payment protocol, shown in Figure 4.10, $\mathcal{U}$ provides $\mathcal{S}$ with $A_1, A_2, A, B, (z, a, b, r)$. $\mathcal{S}$ verifies some properties of A, namely

$$A \overset{?}{=} A_1 A_2,$$

$$A \overset{?}{\neq} 1\ ^{11},$$

and $\mathcal{B}$'s signature $sig(A, B) = (z, a, b, r)$ by checking

$$g^r \overset{?}{=} a h^{\mathcal{H}(A||B||z||a||b)} \text{ and } A^r \overset{?}{=} b z^{\mathcal{H}(A||B||z||a||b)}.$$

Then, a challenge d is computed based on the coin and on information specific to this payment ($I_{\mathcal{S}}$, date/time). $\mathcal{U}$ responds with (r_1, r_2), and $\mathcal{S}$ verifies the correctness of this response.

[11] Otherwise, $\mathcal{U}$ might have chosen $\mathsf{s} = 0$ during withdrawal, thus being able to prevent double-spender identification by responding $r_1 := 0 + x_1$, $r_2 := 0 + x_2$ in the payment protocol.

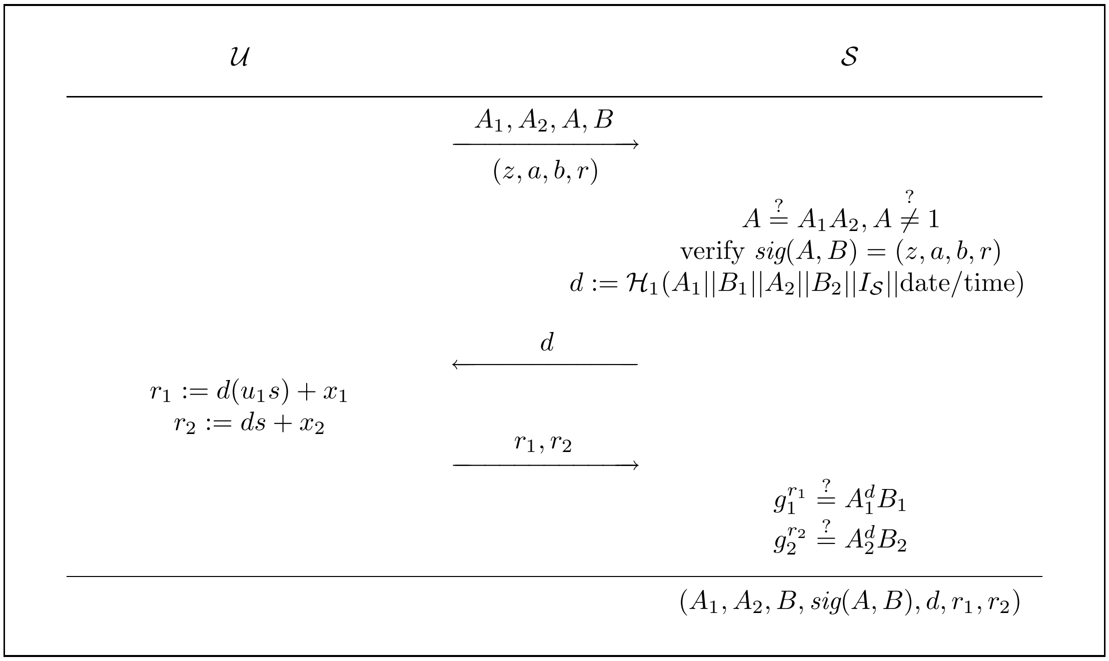

$\mathcal{U}$
$\mathcal{S}$
A_1, A_2, A, B
(z, a, b, r)
$A \overset{?}{=} A_1A_2, A \overset{?}{\neq} 1$
verify $sig(A, B) = (z, a, b, r)$
$d := \mathcal{H}_1(A_1||B_1||A_2||B_2||I_{\mathcal{S}}||\text{date/time})$
d
$r_1 := d(u_1s) + x_1$
$r_2 := ds + x_2$
r_1, r_2
$g_1^{r_1} \overset{?}{=} A_1^d B_1$
$g_2^{r_2} \overset{?}{=} A_2^d B_2$
$(A_1, A_2, B, sig(A, B), d, r_1, r_2)$

Figure 4.10: Payment in the system from [FTY96], part 1.

In the second part of the protocol, shown in Figure 4.11, $\mathcal{U}$ gives $\mathcal{S}$ the pair $(\bar{A}, N)$ and $\mathcal{B}'s$ signature $(\zeta, \alpha, \beta, \rho)$ on it [12]; $\mathcal{S}$ verifies it by checking

$$g^{\rho} \overset{?}{=} \alpha {h'}^{\mathcal{H}(\bar{A}||N||\zeta||\alpha||\beta)} \text{ and } \bar{A}^{\rho} \overset{?}{=} \beta \zeta^{\mathcal{H}(\bar{A}||N||\zeta||\alpha||\beta)}.$$

Additionally, $\mathcal{S}$ verifies that A_2 is embedded in $\bar{A}$ by checking

$$\bar{A} \overset{?}{=} A_2^N g_1^N.$$

$\mathcal{U}$ computes the ElGamal encryption $enc_{OT} := (D_1, D_2) := (Ig_2^{X_{\mathcal{T}}m}, g_2^m)$ of his identity I and engages in an indirect discourse proof with $\mathcal{S}$ to prove its correct construction. The encryption enc_{OT} makes owner tracing possible.

Double-spender Identification

If a coin is deposited more than once, $\mathcal{B}$ obtains at least two payment transcripts for the same coin. If $I_{\mathcal{S}}$, the date/time, and the response (r_1, r_2) of $\mathcal{U}$ are equal in both transcripts, $\mathcal{S}$ is trying to deposit a coin twice. In the case that the coin was actually double-spent by $\mathcal{U}$, the challenges from payment will be different (otherwise $\mathcal{U}$ would have found a collision for the hash function $\mathcal{H}_1$), and therefore with high probability also the responses (r_1, r_2) and (r_1', r_2'). $\mathcal{B}$ can then compute

$$\frac{r_1 - r_1'}{r_2 - r_2'} = u_1,$$

thus obtaining the account number $I = g_1^{u_1}$ of the double-spender and u_1, which serves as a proof of double-spending.

Anonymity Revocation

The anonymity can be revoked by $\mathcal{T}$ by use of his secret key $X_{\mathcal{T}}$; the two different kinds of anonymity revocation are performed as follows:

[12] Actually, the signature is not on the pair $(\bar{A}; N)$, but on the concatenation $(\bar{A}||N)$.

$\mathcal{U}$		$\mathcal{S}$
		(A_2)
	$\xrightarrow[(\zeta, \alpha, \beta, \rho)]{(\bar{A}, N)}$	
		verify $sig(\bar{A}, N) = (\zeta, \alpha, \beta, \rho)$ $\bar{A} \overset{?}{=} A_2^N g_1^N$
$m \in_R \mathbf{Z}_q$ $D_1 := I g_2^{X_{\mathcal{T}} m}$, $D_2 := g_2^m$ $enc_{OT} := (D_1, D_2)$		
	$\xrightarrow{enc_{OT}}$	
		$D_2 \overset{?}{\neq} 1$ $s_0, s_1, s_2 \in_R \mathbf{Z}_q$ $D' := D_1^{s_0} g_2^{s_1} D_2^{s_2}$ $f_2' := f_2^{s_0} g_2^{s_2}$
	$\xleftarrow{D', f_2'}$	
$V := \mathcal{H}_1(D'^s / f_2'^{ms})$		
	$\xrightarrow{V}$	
		$V \overset{?}{=} \mathcal{H}_1(A_1^{s_0} A_2^{s_1})$
		$(\bar{A}, n_1, n_2, sig(\bar{A}, N), enc_{OT})$

Figure 4.11: Payment in the system from [FTY96], part 2.

- Payment-based (owner tracing): $\mathcal{T}$ is given a pair $(D_1, D_2) = (Ig_2^{X_\mathcal{T} m}, g_2^m)$ observed in a payment; $\mathcal{T}$ computes $D_1/D_2^{X_\mathcal{T}} = I$, and $\mathcal{B}$ searches his account database for I. NB. In the system from [CMS96] the larger withdrawal database has to be searched.

- Withdrawal-based (coin tracing): $\mathcal{T}$ is given a pair $(D'_1, D'_2) = (A_2 g_1^{X_\mathcal{T} m'}, g_1^{m'})$ observed in a withdrawal; $\mathcal{T}$ computes $D'_1/(D'_2)^{X_\mathcal{T}} = A_2$, and A_2 can be put on a blacklist for recognizing the coin when it is spent or deposited. NB. In the system from [CMS96] no additional encryption like (D'_1, D'_2) (and thus no signature on this encryption) is needed.

Security for $\mathcal{U}$

In the system from [Bra93], $\mathcal{U}$'s anonymity was unconditional. We know that in systems with passive trustee it can only be computational. We'll now examine the additions to the system from [Bra93] and see to what extent they compromise $\mathcal{U}$'s anonymity:

- The splitting of $A = g_1^{u_1 s} g_2^s$ into $A_1 = g_1^{u_1 s}$, $A_2 = g_2^s$ during payment already reduces the anonymity from unconditional to computational: computing $\log_{g_2} A_2$ will yield s, and then $u_1 = \log_{g_1^s} A_1$. To be able to trace, $\mathcal{B}$ doesn't actually need as much as the capability to compute discrete logarithms; it suffices if $\mathcal{B}$ can solve a problem that was called "matching Diffie-Hellman" problem in [FTY96]. It consists of the following: given $(g^{a_0}, g^{a_0 b_0})$, $(g^{a_1}, g^{a_1 b_1})$, and g^{b_r}, $g^{b_{1-r}}$, find $r \in \{0, 1\}$ with probability significantly better than 1/2 [13]. [FTY96] contains a reduction of the matching Diffie-Hellman problem to the problem of linking withdrawal and payment transcripts.

- $\mathcal{U}$ gives the additional values $A'_1, A'_2, A', \bar{A}$ to $\mathcal{B}$ or $\mathcal{S}$ or both; they are linked to his identity, but are all blinded by raising them to the power of randomly chosen blinding factors. If the values A'_1, A'_2 were omitted, A' and $\bar{A}$ would be unconditionally unlinkable (we would then have the basic restrictive blind signature from [Bra93]), but the splitting of A' into A'_1, A'_2 makes the unlinkability computational, in perfect analogy to the splitting of A into A_1, A_2 described above.

- $\mathcal{U}$ proves that he knows how to represent A'_1, A'_2; this is done by the Schnorr identification protocol, which is believed to be safe, although not perfect zero-knowledge.

- $\mathcal{U}$ computes the encryptions enc_{OT}, enc_{CT}; these are ElGamal encryptions and as such believed to be computationally secure, i.e. given only two plaintexts and their encryptions, it is infeasible to tell which plaintext belongs to which encryption with probability significantly better than 1/2.

- In the third part of the withdrawal protocol, $\mathcal{U}$ and $\mathcal{B}$ engage in a further blind signature protocol; its blindness guarantees that $\mathcal{U}$'s anonymity isn't compromised by it.

- $\mathcal{U}$ conducts two indirect discourse proofs, one with $\mathcal{B}$, one with $\mathcal{S}$. The proofs are zero-knowledge; we'll now show this for the proof executed with $\mathcal{S}$:

  $\mathcal{S}$ can simulate the indirect discourse proof protocol if he is allowed to choose (D_1, D_2): this encryption then encodes the identity of some $\mathcal{U}'$ for a coin consisting of A'_1, A'_2. The security of the ElGamal encryption guarantees that encryptions of different plaintexts are indistinguishable. The values s_0, s_1, s_2 are chosen randomly, and V is computed as $\mathcal{H}_1\big((A'_1)^{s_0}(A'_2)^{s_1}\big)$. $\mathcal{H}_1$ being seen as a random oracle, the distribution of the simulated values is indistinguishable from the one of the real protocol.

$\mathcal{U}$ can't be falsely accused of overspending, as this would require forging a payment transcript; two different payment transcripts would reveal the representation of A_1 and A_2, thus solving an instance of the representation problem.

If $\mathcal{U}$ indeed overspends, the representation of A_1, A_2 becomes known to $\mathcal{B}$ at deposit and even to $\mathcal{S}$ during payment, if $\mathcal{U}$ spends a coin twice at the same $\mathcal{S}$. Anyone knowing this representation could then do further overspending.

[13] Better than $1{=}2 + 1{=}k^c$ for any constant c for large enough k, where k is a security parameter.

Security for $\mathcal{B}$ and $\mathcal{S}$

The information given to $\mathcal{B}$ and $\mathcal{S}$ in this scheme is a superset of that shown in the scheme from [Bra93]; $\mathcal{B}$ and $\mathcal{S}$ do not transmit additional information, and the verifications check everything that was checked in [Bra93]. The additional information obtained by $\mathcal{S}$ doesn't allow him to forge coins or overspend a coin received in payment. Thus, the scheme is as secure for $\mathcal{B}$ and $\mathcal{S}$ as the one from [Bra93]. As we stated before, this scheme hasn't been broken, but its security can't be proven.

Security for $\mathcal{T}$

For $\mathcal{T}$ being able to trace, the encryptions enc_{CT}, enc_{OT} have to be computed correctly. This fact should be proven by the indirect discourse proofs. $\mathcal{T}$ has to trust $\mathcal{B}$ and $\mathcal{S}$ that they perform the proof protocols with $\mathcal{U}$ correctly, i.e. that they don't cooperate with a cheating $\mathcal{U}$ or cheat themselves. We'll take a look at the soundness of the proof between $\mathcal{U}$ and $\mathcal{S}$ about enc_{OT}. Our considerations obviously also apply to the proof between $\mathcal{U}$ and $\mathcal{B}$.

We'll first present the argumentation from [FTY96] and then proceed with our own thoughts.

Soundness of the indirect discourse proof as shown in Section 5.1 of [FTY96]

$\mathcal{U}$ wants to convince $\mathcal{S}$ of the correct construction of $enc_{OT} := (D_1, D_2)$. For simplicity, $\mathcal{U}$ is assumed to be able to compute discrete logarithms.

In the indirect discourse proof, let a cheating user $\mathcal{U}$ try to find a V that $\mathcal{S}$ will accept. After receiving D', f_2' from $\mathcal{S}$, $\mathcal{U}$ might try to find the random values s_0, s_1, s_2, as this would allow him to compute $A_1^{s_0} A_2^{s_1}$. For any (guessed) $s_0' \in \mathbf{Z}_q$ there exist unique $s_1', s_2' \in \mathbf{Z}_q$ such that

$$f_2' = f_2^{s_0'} g_2^{s_2'} \text{ and } D' = D_1^{s_0'} g_2^{s_1'} D_2^{s_2'}.$$

Therefore all guessed $s_0' \in \mathbf{Z}_q$ have equal probability and even a computationally unlimited $\mathcal{U}$ can't guess the correct s_0 with probability greater than $1/q$.

In [FTY96], $\mathcal{U}$ is assumed to cheat by choosing $D_1 := g_1^{u_1'} g_2^{mX_\mathcal{T}}$ with $u_1' \neq u_1$, i.e. by encrypting not his own, but some other identity. NB.: $\mathcal{U}$ could also cheat in other ways, e.g. by encrypting his real identity, but not with $\mathcal{T}$'s public key. This will be discussed below ("further investigation").

Now let's assume that during an execution of the protocol (with (D_1, D_2) incorrectly constructed), $\mathcal{U}$ could compute a V that $\mathcal{S}$ accepts, i.e. for which the verification equation $V = \mathcal{H}_1(A_1^{s_0} A_2^{s_1})$ holds. Then, without loss of generality, he could find all values $\tilde{A}$ satisfying $V = \mathcal{H}_1(\tilde{A})$. With Z defined as

$$Z := (D')^s / (f_2')^{ms} = (g_1^{u_1' s})^{s_0} A_2^{s_1},$$

$\mathcal{U}$ would then perform the following computation for each of these $\tilde{A}$:

$$\begin{array}{rcl} Z' & := & \tilde{A}/Z = g_1^{(u' - u_1')ss_0} \quad \text{(for some } u'\text{)} \\ s_0' & := & \log_Y Z' \quad \text{ with } Y := g_1^{(u_1 - u_1')s}. \end{array}$$

Notice that for one of the $\tilde{A}$, $u' = u_1$ holds, and the s_0' computed for this $\tilde{A}$ is indeed s_0. Since with extremely small probability $|\{\tilde{A} \in \mathbf{Z}_q | V = \mathcal{H}_1(\tilde{A})\}| = q$, i.e. since there are almost certainly less than q values $\tilde{A}$ and therefore less than q corresponding values s_0', $\mathcal{U}$ could randomly choose from these q values and get s_0 with probability strictly greater than $1/q$. This is a contradiction to the maximum probability of success for guessing s_0 stated above. Thus, we conclude that it's infeasible for $\mathcal{U}$ to find a correct V if he chooses

$$D_1 := g_1^{u_1'} g_2^{mX_\mathcal{T}} \quad \text{ with } u_1' \neq u_1.$$

Further Investigation on the Security of the Indirect Discourse Proofs

In the proof of security of the indirect discourse proof in Section 5.1 of [FTY96], it is only shown that $\mathcal{U}$ can't cheat by encrypting a false identity in (D_1, D_2). However, it remains unclear why $\mathcal{S}$ should be convinced of more than that, i.e. how can $\mathcal{S}$ be sure that $\mathcal{U}$ isn't cheating in some other way? For complete security, $\mathcal{S}$ must have certainty over the correct construction of (D_1, D_2). To examine why the presented protocol might satisfy this requirement, we'll take a closer look at the indirect discourse proof that should prove the correctness of (D_1, D_2).

$$\begin{array}{ccc}
\mathcal{U} & & \mathcal{S} \\
A_1 := g_1^{u_1 s}, A_2 := g_2^s & & \\
\hline
m \in_R \mathbf{Z}_q & & \\
D_1 := I g_2^{X_{\mathcal{T}} m} = g_1^{u_1} g_2^{X_{\mathcal{T}} m} & & \\
D_2 := g_2^m & & \\
 & \xrightarrow{D_1, D_2} & \\
 & & D_2 \overset{?}{\neq} 1 \\
 & & s_0, s_1, s_2 \in_R \mathbf{Z}_q \\
 & & D' := D_1^{s_0} g_2^{s_1} D_2^{s_2} \\
 & & f_2' := f_2^{s_0} g_2^{s_2} = g_2^{X_{\mathcal{T}} s_0} g_2^{s_2} \\
 & \xleftarrow{D', f_2'} & \\
\begin{aligned}
V \;&:=\; \mathcal{H}_1\big((D')^s/(f_2')^{ms}\big) \\
&=\; \mathcal{H}_1\left(\frac{D_1^{s_0 s} g_2^{s_1 s} D_2^{s_2 s}}{f_2^{s_0 m s} g_2^{s_2 m s}}\right) \\
&=\; \mathcal{H}_1\left(\frac{g_1^{u_1 s_0 s} g_2^{X_{\mathcal{T}} m s_0 s} g_2^{s_1 s} g_2^{m s_2 s}}{g_2^{X_{\mathcal{T}} s_0 m s} g_2^{s_2 m s}}\right) \\
&=\; \mathcal{H}_1\big((g_1^{u_1 s})^{s_0} (g_2^s)^{s_1}\big) \\
&=\; \mathcal{H}_1(A_1^{s_0} A_2^{s_1})
\end{aligned} & & \\
 & \xrightarrow{V} & \\
 & & V \overset{?}{=} \mathcal{H}_1(A_1^{s_0} A_2^{s_1})
\end{array}$$

Figure 4.12: Analysis of the indirect discourse proof during payment.

In Figure 4.12, the expression $\mathcal{H}_1\big((D')^s/(f_2')^{ms}\big)$ is expanded to show how its basic components actually "sum up" to $\mathcal{H}_1(A_1^{s_0} A_2^{s_1})$. Of course, this happens under the assumption that both participants are honest. However, what possibilities does $\mathcal{U}$ have for cheating? As shown in the previous subsection, guessing s_0, s_1, s_2 is infeasible. We assume that $\mathcal{U}$ can compute discrete logarithms and thus obtain α with $g_2 = g_1^{\alpha}$. Computing the discrete logarithms of D', f_2', and $D'^s/f_2'^{ms} = (g_1^{u_1' s})^{s_0} A_2^{s_1}$ would yield

$$\begin{aligned}
\log_{g_1} D' \;&=\; (u_1 + \alpha X_{\mathcal{T}} m) s_0 + \alpha s_1 + \alpha m s_2 \\
\log_{g_1} f_2' \;&=\; \alpha X_{\mathcal{T}} s_0 + \alpha s_2 \\
\log_{g_1} (g_1^{u_1' s})^{s_0} A_2^{s_1} \;&=\; u_1' s s_0 + \alpha s s_1.
\end{aligned}$$

As can be seen, there is no way of getting to know s_0, s_1, s_2 like this, either. As these random factors remain unknown, $\mathcal{U}$ has no possibility to manipulate the values D', f_2' obtained from $\mathcal{S}$ in way that will lead to validity of the verification equation.

Substituting some $s' \neq s$ for s in the calculation of V would allow $\mathcal{U}$ to "hide" a false u_1' used for constructing D_1 (e.g. by choosing $s' := u_1/u_1'$), but would at the same time cause $A_2^{s_1} \neq g_2^{s' s_1}$. Similarly, substituting $m' \neq m$ for m at some point (when computing (D_1, D_2) or V or both) won't yield a valid V for an incorrectly constructed (D_1, D_2).

Roughly speaking, the obstacle for a cheating $\mathcal{U}$ is that for successfully manipulating the values used in the calculation of V, $\mathcal{U}$ needs at least one of the randomly chosen values s_0, s_1, s_2. As long as $\mathcal{S}$ doesn't reveal any of them, $\mathcal{U}$ can only guess one of them with probability $1/q$. Thus, the argumentation from [FTY96] presented in the previous subsection seems reasonable even if we allow $\mathcal{U}$ not only to encrypt a false u_1', but to compute (D_1, D_2) in any way he likes. This is clear if one thinks that any pair (D_1, D_2) can be seen as an encryption of some false identity with $\mathcal{T}$'s public key, i.e. for all $(D_1, D_2) \in (\mathbf{Z}_q \times \mathbf{Z}_q)$ there exist u_1', m such that

$$D_1 = g_1^{u_1'} f_2^m, \quad D_2 = g_2^m.$$

4.3.4 The Off-line Systems from [FTY98]

In [FTY98], an extended proof of equality of logarithms was used to construct indirect discourse proofs. These indirect discourse proofs are, in contrast to those from [FTY96], noninteractive. It was first shown how to use them to add anonymity revocation to an existing anonymous system ([Bra93]); afterwards, a proprietary, more efficient system employing extended proofs of equality of logarithms and without indirect discourse proofs was presented.

$\mathcal{U}$'s anonymity was claimed to be provable under the Decision Diffie-Hellman assumption for both approaches, while security under all other aspects should be the same as in the underlying system from [Bra93].

If consulting [FTY98], the following should be noticed:

- the variables Q and P should be replaced by q and p,
- in the protocol in Figure 2 on page 6, s_3 isn't needed,
- throughout Section 4.1, u actually means u_1,
- in the protocol in Figure 3 on Page 8, r_3 must be computed as $r_3 := s_3 - cxu_1$ for the protocol to work,
- the definition of V_2 at the bottom of Page 8 should read

  $$V_2 = \mathrm{IndPrf}[(\bar{A}, g_2 g_4^t), g_1, (\bar{A}, g_1, D_1|f_2), g_2 g_4^t],$$

- on Page 11, all occurrences of t^{-1} should be replaced by $-t$.

The extended proof of equality of logarithms is shown in Figure 4.13; soundness and zero-knowledge of it were discussed in [FTY98] (where the authors called "correctness" what we refer to as soundness); it was shown that the prover can't successfully cheat unless he is able to compute discrete logarithms.

For generators a, b, G_1, G_2 and values A, B, it proves that $\mathcal{P}$ knows values x, v, w such that $A = a^x G_1^v$, $B = b^x G_2^w$. Thus, it is actually a proof of knowledge of representation of A, B and of equality of one exponent. We'll refer to it as $\mathit{Proof}_{REP+EQLOG}((A, a), G_1, (B, b), G_2)$. We're thus almost completely retaining the notation from [FTY98], which is not very intuitive; we chose to do so to ease using [FTY98] as a reference. However, the meaning of each proof is explained in clear words.

The indirect discourse proof is shown in Figure 4.14; for generators a, b, G_1, G_2, G_3 and values A, B, C it proves that $\mathcal{P}$ knows values x, v, w, u such that $A = a^x G_1^v$, $B = C^x G_2^z G_3^t = b^{ex} G_2^w$, $C = b^e G_3^u$ for some values z, t, e.

As discussion of soundness and zero knowledge of the proof were omitted in [FTY98], they are dealt with here. When trying to show that the indirect discourse proof is correct and sound (i.e. that it actually proves what it should prove), some problems were encountered: as will be shown below, a knowledge extractor (see Subsection 2.7.2) can't obtain the prover's secret values. Nevertheless, successfully cheating in the proof protocol would result in the ability to compute discrete logarithms. As this is regarded as

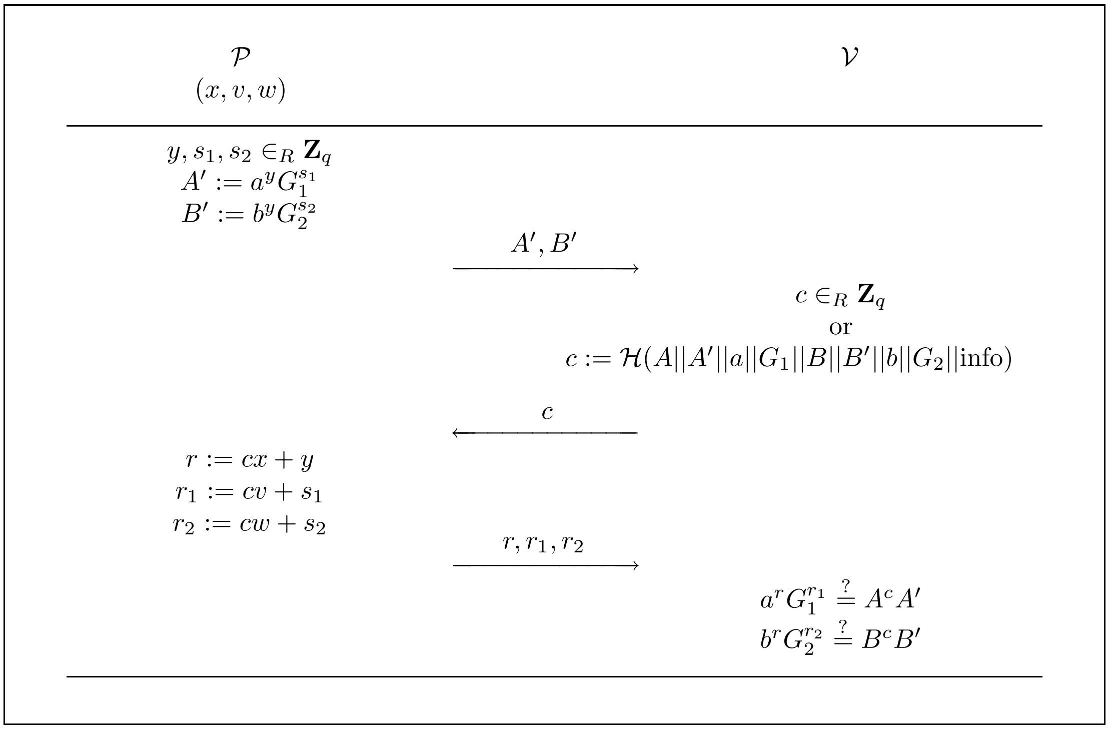

$\mathcal{P}$
(x, v, w)
$\mathcal{V}$
$y, s_1, s_2 \in_R \mathbf{Z}_q$
$A' := a^y G_1^{s_1}$
$B' := b^y G_2^{s_2}$
$\xrightarrow{A', B'}$
$c \in_R \mathbf{Z}_q$
or
$c := \mathcal{H}(A||A'||a||G_1||B||B'||b||G_2||\text{info})$
$\xleftarrow{c}$
$r := cx + y$
$r_1 := cv + s_1$
$r_2 := cw + s_2$
$\xrightarrow{r, r_1, r_2}$
$a^r G_1^{r_1} \overset{?}{=} A^c A'$
$b^r G_2^{r_2} \overset{?}{=} B^c B'$

Figure 4.13: Proof of equality of logarithms from [FTY98].

infeasible, it seems plausible that although the indirect discourse proof of [FTY98] isn't a knowledge proof in the strict sense, it is secure if computing discrete logarithms is infeasible.

The zero-knowledge property of the indirect discourse proof is shown by demonstrating how the verifier could simulate the proof.

Soundness of the Indirect Discourse Proof

If a proof of knowledge actually proves knowledge of secrets, a knowledge extractor $\mathcal{E}$ should be able to extract these secrets by execution of the proof protocol: In the concrete case of the indirect discourse proofs as presented in Figure 4.14, $\mathcal{E}$ begins the protocol with the prover and obtains upon his challenge c the response (r, r_1, r_2, r_3). At this point, $\mathcal{E}$ resets the prover to the step of the protocol where the values A', B' have just been sent. A different challenge c' is passed to $\mathcal{P}$, and $\mathcal{E}$ obtains the corresponding response (r', r_1', r_2', r_3'). If the prover is honest, $\mathcal{E}$ has only to perform the following computations:

$$\begin{aligned}
r - r' = cx - c'x = (c - c')x \quad &\Rightarrow \quad x = \frac{r - r'}{c - c'} \\
r_1 - r_1' = cv - c'v = (c - c')v \quad &\Rightarrow \quad v = \frac{r_1 - r_1'}{c - c'} \\
r_2 - r_2' = cw - c'w = (c - c')w \quad &\Rightarrow \quad w = \frac{r_2 - r_2'}{c - c'} \\
r_3 - r_3' = -cxu + c'xu = (c' - c)xu = (r - r')u \quad &\Rightarrow \quad u = \frac{r_3 - r_3'}{r - r'}.
\end{aligned}$$

As can be seen, the right-hand sides of the equations for x, v, w, u contain only values known to $\mathcal{E}$ and thus $\mathcal{E}$ obtains the prover's secrets.

However, this result shows only the correctness of the protocol, i.e. if executed by an honest prover, the protocol will reveal the (correct) secret values to a knowledge extractor $\mathcal{E}$. When allowing the prover to cheat, $\mathcal{E}$ can't pretend that the responses were actually computed according to the protocol. The only

$$\begin{array}{ccc}
\mathcal{P} & & \mathcal{V} \\
(x, v, w, u) & & \\
\hline
y, s_1, s_2, s_3 \in_R \mathbf{Z}_q & & \\
A' := a^y G_1^{s_1} & & \\
B' := C^y G_2^{s_2} G_3^{s_3} & & \\
 & \xrightarrow{A', B'} & \\
 & & c \in_R \mathbf{Z}_q \\
 & \xleftarrow{c} & \\
r := cx + y & & \\
r_1 := cv + s_1 & & \\
r_2 := cw + s_2 & & \\
r_3 := s_3 - cxu & & \\
 & \xrightarrow{r, r_1, r_2, r_3} & \\
 & & a^r G_1^{r_1} \overset{?}{=} A^c A' \\
 & & C^r G_2^{r_2} G_3^{r_3} \overset{?}{=} B^c B' \\
\hline
\end{array}$$

Figure 4.14: Indirect discourse proof from [FTY98].

thing he knows is that the verification equations hold:

$$\begin{aligned} a^r G_1^{r_1} &= A^c A' \\ C^r G_2^{r_2} G_3^{r_3} &= B^c B'. \end{aligned}$$

For simplicity, let's assume that the knowledge extractor $\mathcal{E}$ is able to compute discrete logarithms. Therefore in the following, for reasons of legibility, we'll transform all values to powers of a generator g, i.e. instead of writing a we'll be writing $g^{\widehat{a}}$ etc. (where $\widehat{a} := \log_g a$). Thus, the verification equations become

$$g^{\widehat{a}r} g^{\widehat{G_1}r_1} = g^{\widehat{A}c} g^{\widehat{A'}} \quad \Rightarrow \quad \widehat{a}r + \widehat{G_1}r_1 = \widehat{A}c + \widehat{A'} \tag{4.1}$$

$$g^{\widehat{C}r} g^{\widehat{G_2}r_2} g^{\widehat{G_3}r_3} = g^{\widehat{B}c} g^{\widehat{B'}} \quad \Rightarrow \quad \widehat{C}r + \widehat{G_2}r_2 + \widehat{G_3}r_3 = \widehat{B}c + \widehat{B'}. \tag{4.2}$$

Now, if the indirect discourse proof indeed proves that $A = a^x G_1^v$ and $B = C^x G_2^z G_3^t = b^{ex} G_2^w$, $\mathcal{E}$ should be able to compute the secret values by applying the equations

$$\widehat{A} = \widehat{a}x + \widehat{G_1}v \tag{4.3}$$

$$\widehat{B} = \widehat{b}ex + \widehat{G_2}w \tag{4.4}$$

$$\widehat{C} = \widehat{b}e + \widehat{G_3}u. \tag{4.5}$$

Subtracting the equation

$$\widehat{a}r' + \widehat{G_1}r_1' = \widehat{A}c' + \widehat{A'} \tag{4.6}$$

(derived from the second challenge/response-pair) from Equation 4.1 and solving the result gives us

$$\widehat{A} = \frac{\widehat{a}(r - r') + \widehat{G_1}(r_1 - r_1')}{c - c'} \tag{4.7}$$

Applying Equation 4.3 to this yields

$$\begin{aligned} \widehat{a}x + \widehat{G_1}v &= \frac{\widehat{a}(r-r') + \widehat{G_1}(r_1 - r_1')}{c-c'} \\ \Rightarrow \quad x &= \frac{\widehat{a}(r-r') + \widehat{G_1}(r_1 - r_1')}{\widehat{a}(c-c')} - \frac{\widehat{G_1}v}{\widehat{a}} \end{aligned} \tag{4.8}$$

Hence, for each v there exists a unique x that fulfills Equation 4.8. By examining Equation 4.2 and the other equations provided, one can see that for each of the pairs (v, x), suitable values w, e, u, z, t can be found. Suitable means that with these values, both the verification equations and the condition to be proven by the indirect discourse proof hold. Thus, $\mathcal{E}$ can't obtain the prover's actual secret values but by guessing at least one of them.

One might hope that more information could be gained by sending more than just two challenges. However, this leads only to more equations of the form of Equation 4.7, with only the number of primes ($'$) varying. The left-hand side would remain the same, leading to something not different enough from Equation 4.8.

The way the authors of [FTY98] argued that their indirect discourse proofs nevertheless are sound was by requiring the prover not to know the relative discrete logarithms of the generators a, G_1, b, G_2. Now, if the prover is able to respond to two challenges, then he can also compute the correct representations of A, B, C. As the challenges are chosen at random, the probability of the prover being able to respond correctly even to a single challenge without knowing the representations is negligible.

However, this actually only proves that such representations exist, but it is not said that they are the ones used by the prover. This can be derived only from the fact that the prover is said not to know the relative discrete logarithms of a, G_1, b, G_2. If the prover knew other representations of A, B, C, he could compute these logarithms:

Let's say that a cheating prover gives a response that an honest prover would have computed from the representation $A = a^x G_1^v$, but the cheating prover knows some other representation $A = a^{x'} G_1^{v'}$. Now the following equations hold:

$$\begin{aligned} a^x G_1^v &= a^{x'} G_1^{v'} \\ \Rightarrow \quad G_1^{v-v'} &= a^{x'-x} \\ \Rightarrow \quad \log_a G_1^{v-v'} &= \log_a a^{x'-x} \\ \Rightarrow \quad (v-v') log_a G_1 &= x' - x \\ \Rightarrow \quad log_a G_1 &= \frac{x'-x}{v-v'}. \end{aligned}$$

Analogous equations exist for B, C. Thus, if the cheating prover knows more than one representation of A, B, C, he can compute the relative discrete logarithms of a, G_1, b, G_2. Then he would be able to compute any discrete logarithm by replacing a, G_1, b, G_2. As this is believed to be infeasible, we can assume that for being able to respond correctly, the prover has to know a representation fulfilling the condition to be proven.

Zero Knowledge

It is sufficient to show that the proof could be simulated in a way indistinguishable to $\mathcal{V}$. To this end, a simulator $\mathcal{S}$ would first randomly choose the challenge c and the response (r, r_1, r_2, r_3). The values A', B' would then be computed as

$$\begin{aligned} A' &:= a^r G_1^{r_1} A^{-c} \\ B' &:= C^r G_2^{r_2} G_3^{r_3} B^{-c}. \end{aligned}$$

With these values, the verification equations clearly hold. Moreover, the distribution of the values constituting $\mathcal{V}$'s view is the same as of those resulting from an execution of the protocol. Thus, simulated views are indistinguishable from real ones.

If the proof is made noninteractive by letting $\mathcal{P}$ compute c as

$$c := \mathcal{H}(A||B||C||A'||B'||a||b||G_1||G_2||G_3||\text{date/time}||\text{info}),$$

then during a simulation of the proof, the hash function $\mathcal{H}$ (which is seen as a random oracle) has to be modified to output c on input $A||B||C||A'||B'||a||b||G_1||G_2||G_3||$date/time$||$info. The resulting modified $\mathcal{H}$ can't be distinguished from a real random oracle, since c is a random value and since we can assume that the probability that $A||B||C||A'||B'||a||b||G_1||G_2||G_3||$date/time$||$info has already been asked before [14] is negligible.

4.3.5 Modular Addition of Owner Tracing from [FTY98]

In the following, the modular addition from [FTY98] of owner tracing to Brands' system [Bra93] is presented. It consists of an extension to the payment phase: $\mathcal{U}$ provides $\mathcal{S}$ with an ElGamal encryption of his identity. The encryption has to be linked to the coin; otherwise, $\mathcal{S}$ would have to be trusted to store coin-encryption pairs correctly.

We'd like to stress that the following is an *addition* to the payment protocol from [Bra93]; in [FTY98] this addition was called "the new payment protocol". It seems that the authors forgot that the signature of $\mathcal{B}$ on the coin still has to be checked by $\mathcal{S}$ during payment. However, the sending of $A, B, sig(A, B)$ to $\mathcal{S}$ and the verification of the signature is all that's missing, i.e. the challenge and response from the payment protocol from [Bra93] as shown in Figure 3.4 on Page 31 can be omitted. This is due to the fact that equivalent responses are given already in the proofs V_1, V_2 presented below.

Payment

During payment, $\mathcal{U}$ computes an ElGamal encryption (D_1, D_2) of I and engages in two proofs V_1, V_2. The first of them, presented in Figure 4.15, is an extended proof of equality of logarithms which shows that $D_1 = g_1^x f_2^m$, $D_2 = g_2^m$ for some x, m. Notice that computing powers of 1 is only done to maintain conformity with the general protocol for proving equality of logarithms. The verification equations for a correctly computed V_1 hold:

$$\begin{aligned}
f_2^r g_1^{r_1} &= f_2^{cm+y} g_1^{cx+s_1} \\
&= f_2^{cm} g_1^{cx} f_2^y g_1^{s_1} \\
&= (g_2^{X_\mathcal{T} m} g_1^x)^c f_2^y g_1^{s_1} \\
&= D_1^c A'
\end{aligned}$$

$$\begin{aligned}
g_2^r 1^{r_2} &= g_2^{cm+y} \\
&= (g_2^m)^c g_2^y \\
&= D_2^c B'.
\end{aligned}$$

Strangely, for the indirect discourse proof V_2 the coin was without further explanation defined to be $\bar{A} := g_1^{u_1 s}(g_2 g_4^t)^s$, thus being different from $A := g_1^{u_1 s} g_2^s$ from [Bra93]. This modification means that the generator g_2 is replaced by a new generator $g_2 g_4^t$. Obviously, this has to be done for the whole withdrawal protocol from [Bra93].

Now, let's examine the indirect discourse proof V_2 shown in Figure 4.16; in addition to redefining the coin, the random values y, s_1 are replaced by x_2, x_1 and thus, A' is replaced by $B = g_1^{x_1}(g_2 g_4^t)^{x_2}$; these values are the same that were chosen randomly during withdrawal. This is done to let double-spending result in an identification of $\mathcal{U}$ by $\mathcal{B}$ without the help of the trustee.

The indirect discourse proof V_2 shows that for the value x from V_1, the equivalence $xs \equiv u_1 s \pmod q$ holds; this implies $x \equiv u_1 \pmod q$ and therefore proves $D_1 = I g_2^{X_\mathcal{T}}$. The relation $xs \equiv u_1 s \pmod q$ is derived from the following relations proven by V_2 for some values $\tilde{x}, \tilde{e}, \tilde{u}, \tilde{v}, \tilde{w}, \tilde{z}, \tilde{t}$ (replacing the corresponding values x, e, u, v, w, z, t from Figure 4.14 to avoid ambiguities):

[14] The response of $\mathcal{H}$ to this previous query would probably not have been c, and thus $\mathcal{V}$ would notice that $\mathcal{H}$ has been modified.

Prove that $D_1 = g_1^x g_2^{X_{\mathcal{T}} m}$, $D_2 = g_2^m$ for some x, m:

$\mathcal{U}$ $(D_1, D_2, f_2 = g_2^{X_{\mathcal{T}}})$		$\mathcal{S}$ $(D_1, D_2, f_2 = g_2^{X_{\mathcal{T}}})$
$y, s_1, s_2 \in_R \mathbf{Z}_q$ $A' := f_2^y g_1^{s_1}$ $B' := g_2^y 1^{s_2}$		
	$\xrightarrow{A', B'}$	
		$c \in_R \mathbf{Z}_q$
	$\xleftarrow{c}$	
$r := cm + y$ $r_1 := cx + s_1$ $r_2 := c + s_2$		
	$\xrightarrow{r, r_1, r_2}$	
		$f_2^r g_1^{r_1} \overset{?}{=} D_1^c A'$ $g_2^r 1^{r_2} \overset{?}{=} D_2^c B'$
		(c, r, r_1, r_2)

Figure 4.15: Payment in modular owner tracing from [FTY98], V_1.

Prove that $\bar{A} = (g_2 g_4^t)^s g_1^v$, $\bar{A} = D_1^s (g_2 g_4^t)^z f_2^p = g_1^{u_1 s} (g_2 g_4^t)^s$:

$\mathcal{U}$ $(\bar{A}, D_1, D_2, f_2 = g_2^{X_{\mathcal{T}}})$		$\mathcal{S}$ $(\bar{A}, D_1, D_2, f_2 = g_2^{X_{\mathcal{T}}})$
$s_2, s_3 \in_R \mathbf{Z}_q$ $B = g_1^{x_1} (g_2 g_4^t)^{x_2}$ $B' := D_1^{x_2} (g_2 g_4^t)^{s_2} f_2^{s_3}$		
	$\xrightarrow{B, B'}$	
		$c \in_R \mathbf{Z}_q$
	$\xleftarrow{c}$	
$r := cs + x_2$ $r_1 := cu_1 s + x_1$ $r_2 := cs + s_2$ $r_3 := s_3 - csm$		
	$\xrightarrow{r, r_1, r_2, r_3}$	
		$(g_2 g_4^t)^r g_1^{r_1} \overset{?}{=} \bar{A}^c B$ $D_1^r (g_2 g_4^t)^{r_2} f_2^{r_3} \overset{?}{=} \bar{A}^c B'$
		(c, r, r_1, r_2, r_3)

Figure 4.16: Payment in modular owner tracing from [FTY98], V_2.

- $\bar{A} = (g_2 g_4^t)^{\tilde{x}} g_1^{\tilde{v}}$,
- $\bar{A} = g_1^{\tilde{e}\tilde{x}} (g_2 g_4^t)^{\tilde{w}} = D_1^{\tilde{x}} (g_2 g_4^t)^{\tilde{z}} f_2^{\tilde{t}}$,
- $D_1 = g_1^{\tilde{e}} f_2^{\tilde{u}}$.

We now replace $\tilde{e}$ by x and $\tilde{u}$ by m from V_1, and $\tilde{x}$ by s and $\tilde{v}$ by $u_1 s$ due to the definition of $\bar{A}$ and to the restrictiveness of the blind signature protocol, and obtain

- $\bar{A} = (g_2 g_4^t)^{s} g_1^{u_1 s}$,
- $\bar{A} = g_1^{xs} (g_2 g_4^t)^{\tilde{w}} = D_1^{s} (g_2 g_4^t)^{\tilde{z}} f_2^{\tilde{t}}$,
- $D_1 = g_1^{x} f_2^{m}$.

Assuming that $\mathcal{U}$ knows only one representation of $\bar{A}$ with respect to $(g_1, g_2 g_4^t)$, we then have $xs \equiv u_1 s \pmod{q}$ and in consequence $D_1 = g_1^{u_1} f_2^m$.

Now it should be clear why an extended proof of equality of logarithms wouldn't be sufficient to show the correct structure of (D_1, D_2); the indirect discourse proof V_2 proves the more complex relation to $\bar{A}$.

For a correctly computed proof V_2 the verification equations hold:

$$\begin{aligned}
(g_2 g_4^t)^r g_1^{r_1} &= (g_2 g_4^t)^{cs+x_2} g_1^{cu_1 s + x_1} \\
&= (g_2 g_4^t)^{cs} g_1^{cu_1 s} (g_2 g_4^t)^{x_2} g_1^{x_1} \\
&= \big((g_2 g_4^t)^s g_1^{u_1 s}\big)^c (g_2 g_4^t)^{x_2} g_1^{x_1} \\
&= \bar{A}^c B
\end{aligned}$$

$$\begin{aligned}
D_1^r (g_2 g_4^t)^{r_2} f_2^{r_3} &= D_1^{cs+x_2} (g_2 g_4^t)^{cs+s_2} f_2^{s_3 - csm} \\
&= D_1^{cs} (g_2 g_4^t)^{cs} f_2^{-csm} D_1^{x_2} (g_2 g_4^t)^{s_2} f_2^{s_3} \\
&= \big(D_1^s (g_2 g_4^t)^s f_2^{-sm}\big)^c D_1^{x_2} (g_2 g_4^t)^{s_2} f_2^{s_3} \\
&= \big((g_1^{u_1})^s (f_2^m)^s (g_2 g_4^t)^s f_2^{-sm}\big)^c D_1^{x_2} (g_2 g_4^t)^{s_2} f_2^{s_3} \\
&= \big((g_1^{u_1 s} (g_2 g_4^t)^s\big)^c D_1^{x_2} (g_2 g_4^t)^{s_2} f_2^{s_3} \\
&= \bar{A}^c B'.
\end{aligned}$$

Double-spender Identification

If $\mathcal{U}$ spends the same coin more than once, $\mathcal{B}$ will end up with two payment transcripts containing tuples (c, r, r_1, r_2, r_3), $(c', r', r_1', r_2', r_3')$ resulting from two proofs V_2.

Since the challenges in V_2 are chosen randomly, with high probability $c \neq c'$ (otherwise $\mathcal{S}$ is trying to double-deposit a coin). Since the same values x_1, x_2 are used in both proofs V_2, $\mathcal{B}$ can compute

$$\frac{r_1 - r_1'}{r - r'} = u_1.$$

Anonymity Revocation

Owner tracing is accomplished as follows: $\mathcal{T}$ is given the pair (D_1, D_2) observed in a payment; $\mathcal{T}$ computes $D_1 / D_2^{X_\mathcal{T}} = I$ and thus obtains the account number of $\mathcal{U}$.

Security for $\mathcal{U}$

In [FTY98], a proof for the following fact was sketched: if anonymity is compromised, then the Decision Diffie-Hellman problem can be solved. As proven in [FTY96], $\mathcal{U}$'s anonymity can only be computational.

$\mathcal{U}$ can't be falsely accused of overspending, as this would require forging a payment transcript; doing so without knowing how to represent the coin is as hard as computing discrete logarithms.

If $\mathcal{U}$ indeed overspends, the values u_1, s that serve to represent the coin $\bar{A}$ become known to $\mathcal{B}$ at deposit and even to $\mathcal{S}$ during payment, if $\mathcal{U}$ spends a coin twice at the same $\mathcal{S}$. Anyone knowing these values could then do further overspending.

Security for $\mathcal{B}$ and $\mathcal{S}$

The blind signature protocol used during withdrawal is unchanged from [Bra93] with the exception that the generator g_2 was replaced by $g_2g_4^t$, but this doesn't change security. Hence, the security for $\mathcal{B}$ and $\mathcal{S}$ is as in [Bra93].

Security for $\mathcal{T}$

For $\mathcal{T}$ being able to trace, the encryption (D_1, D_2) has to be computed correctly. This fact should be proven by the proofs V_1, V_2. $\mathcal{T}$ has to trust $\mathcal{S}$ that he performs the proof protocols with $\mathcal{U}$ correctly, i.e. that he doesn't cooperate with a cheating $\mathcal{U}$ or cheat himself. It was shown that $\mathcal{U}$ could cheat by himself during the proofs only if he is able to compute discrete logarithms; hence, under the assumption that this is infeasible, $\mathcal{T}$ will be able to trace.

4.3.6 Modular Addition of Coin Tracing from [FTY98]

In the following, the modular addition from [FTY98] of coin tracing to Brands' system [Bra93] is presented. It consists of extensions to both the withdrawal and the payment phase.

Withdrawal

During withdrawal, a value $I' := Ig_3^{1/s}g_4^t$ is created; an ElGamal encryption $E_1 := (Ig_2g_4^t)^s f_1^m$, $E_2 := g_1^m$ is computed, where $f_1 := g_1^{X_\mathcal{T}}$ is another public key related to $\mathcal{T}$'s secret key $X_\mathcal{T}$. According to [FTY98], the "relationship" of I' to (E_1, E_2) is proven using equality of logarithm proofs. Regrettably, the authors of [FTY98] didn't specify how this is precisely done. We suppose they meant that it is proven that $\mathcal{U}$'s identity I and the blinding factor s are embedded in both I' and E_1. Moreover, one proof might be used to show that (E_1, E_2) is an ElGamal encryption based on the public key f_1. We'll call this latter proof V_1 and define it as

$$V_1 := \mathit{Proof}_{REP+EQLOG}\big((E_1, f_1), Ig_2g_4^t, (E_2, g_1), 1\big).$$

It proves that for values $a, b, c \in \mathbf{Z}_q$ known to $\mathcal{U}$,

$$E_1 = f_1^a (Ig_2g_4^t)^b \quad \text{and} \quad E_2 = g_1^a 1^c.$$

The "relationship" between I' and E_1 is proven by another proof V_2 which we define as

$$V_2 := \mathit{Proof}_{REP+EQLOG}\big((g_3, I'), (I, g_4^t), (E_1, Ig_2g_4^t), f_1\big).$$

It proves that for values $d, e, f, g \in \mathbf{Z}_q$ known to $\mathcal{U}$,

$$g_3 = {I'}^d I^e (g_4^t)^f \quad \text{and} \quad E_1 = (Ig_2g_4^t)^d f_1^g.$$

Assuming that $\mathcal{U}$ knows only one representation of E_1 with respect to $(f_1, Ig_2g_4^t)$ [15], we can assume that $d = b$ and $g = a$ with the values b, a from the proof V_1. Moreover, we can define $d := s$ and $g := m$, thus obtaining

$$g_3 = {I'}^s I^e (g_4^t)^f \quad \text{and} \quad E_1 = (Ig_2g_4^t)^s f_1^m.$$

Now, we have a blinded coin $I'g_2 = Ig_2g_3^{s^{-1}}g_4^t$ which $\mathcal{B}$ can also compute ($\mathcal{B}$ knows I' and g_2). During the blind signature protocol, $\mathcal{U}$ chooses a blinding factor s' and obtains the unblinded coin $A := (I'g_2)^{s'} = I^{s'}g_2^{s'}g_3^{s^{-1}s'}g_4^{ts'}$ and a signature on it.

[15] Otherwise, $\mathcal{U}$ could compute $\log_{f_1} Ig_2g_4^t$.

Payment

In addition to the basic payment protocol, $\mathcal{U}$ sends $\bar{A} := A/g_3$ and $\mathcal{S}$ verifies $A \overset{?}{=} \bar{A}g_3$. Thus, during withdrawal $\mathcal{U}$ is forced to choose $s' := s$.

Anonymity Revocation

Coin tracing is accomplished by $\mathcal{T}$ by decrypting (E_1, E_2) to $(Ig_2g_4^t)^s$, which is the value $\bar{A}$ that is sent to $\mathcal{S}$ at payment. $\bar{A}$ can then be blacklisted.

Security for $\mathcal{U}$

In [FTY98], a proof was sketched that if anonymity is compromised, then the Decision Diffie-Hellman problem can be solved. As proven in [FTY96], $\mathcal{U}$'s anonymity can only be computational.

$\mathcal{U}$ can't be falsely accused of overspending, as this would require forging a payment transcript; as for [Bra93], this is assumed to be infeasible unless one knows how to represent the coin.

If $\mathcal{U}$ indeed overspends, the values u_1, s that serve to represent the coin A become known to $\mathcal{B}$ at deposit and even to $\mathcal{S}$ during payment, if $\mathcal{U}$ spends a coin twice at the same $\mathcal{S}$. Anyone knowing these values could then do further overspending.

Security for $\mathcal{B}$ and $\mathcal{S}$

The blind signature protocol used during withdrawal still contains all elements from [Bra93], hence its security and therefore the security for $\mathcal{B}$ and $\mathcal{S}$ are unchanged.

Security for $\mathcal{T}$

For $\mathcal{T}$ being able to trace, the encryption (E_1, E_2) has to be computed correctly. This fact should be proven by the proofs V_1, V_2. $\mathcal{T}$ has to trust $\mathcal{B}$ that he performs the proof protocols with $\mathcal{U}$ correctly; additionally, $\mathcal{S}$ has to be trusted to accept only coins A for which $\mathcal{U}$ presents a corresponding value $\bar{A}$ such that $A = \bar{A}g_3$. This is no problem, since otherwise $\mathcal{B}$ may refuse the coin at deposit. It was shown that $\mathcal{U}$ could cheat by himself during the proofs only if he is able to compute discrete logarithms; hence, under the assumption that this is infeasible, $\mathcal{T}$ will be able to trace.

4.3.7 "Simplified FOLC" from [FTY98]

"FOLC" stands for fair off-line cash; in the second part of [FTY98], the modular additions of tracing were slightly modified and used to create a scheme that needs less computation, but retains security under the same assumptions.

The idea is to combine $\mathcal{U}$'s identity with a "coin identifier" as in [CMS96] to an unconditional commitment, which is then blindly signed.

System Setup

$\mathcal{B}$ chooses primes p, q with $p = \gamma q + 1$ for a specified small integer γ and the unique subgroup G_q of order q of $\mathbf{Z}_p$ with generators $g, g_1, \ldots, g_4$ such that computing discrete logarithms in G_q is infeasible.

$\mathcal{B}$ also chooses a secret key $X_\mathcal{B} \in_R \mathbf{Z}_q$; there are corresponding public keys

$$h := g^{X_\mathcal{B}}, h_1 := g_1^{X_\mathcal{B}}, h_2 := g_2^{X_\mathcal{B}}, h_3 := g_3^{X_\mathcal{B}}.$$

Additionally, collision-resistant one-way hash functions $\mathcal{H}, \mathcal{H}_0, \ldots$ are defined.

$\mathcal{T}$ chooses a secret key $X_\mathcal{T} \in_R \mathbf{Z}_q$; there are corresponding public keys $f_2 := g_2^{X_\mathcal{T}}, f_3 := g_3^{X_\mathcal{T}}$.

Account Opening

At account opening, $\mathcal{U}$ chooses a secret $u_1 \in \mathbf{Z}_q$ with $g_1^{u_1} g_2 \neq 1$ [16]; $\mathcal{U}$ computes his public identity as $I := g_1^{u_1}$ and proves to $\mathcal{B}$ that he knows the discrete logarithm of I with respect to g_1.

Withdrawal

In the withdrawal protocol shown in Figure 4.17, $\mathcal{U}$ creates an intermediate value I' and an ElGamal encryption (E_1, E_2) of g_2^s based on $\mathcal{T}$'s public key f_3. This encryption allows $\mathcal{T}$ to perform coin tracing. The correct construction of I' and (E_1, E_2) with respect to each other and to $I = g_1^{u_1}$ is proven by two proofs of equality of logarithms V_1, V_2.

$\mathcal{U}$ $(I = g_1^{u_1})$		$\mathcal{B}$
$m, s, t \in_R \mathbf{Z}_q$ $I' := g_1^{u_1 s^{-1}} g_3^{s^{-1}} g_4^t$ $E_1 := g_2^s f_3^m, E_2 := g_3^m$ $V_1 := Proof_{REP+EQLOG}\big((E_1, f_3), g_2, (E_2, g_3), 1\big)$ $V_2 := Proof_{REP+EQLOG}\big((g_3, I'), (g_1, g_4), (E_1, g_2), f_3, (I, I'), (g_3, g_4)\big)$		
	$\xrightarrow{I', E_1, E_2, V_1, V_2}$	
		$E_2 \overset{?}{\neq} 1$ verify V_1, V_2 $w \in_R \mathbf{Z}_q$ $a' := g^w, b' := (I' g_2)^w, b'' := g_4^w$
	$\xleftarrow{a', b', b''}$	
$A := (I' g_2 g_4^{-t})^s = g_1^{u_1} g_2^s g_3$ $z := h_1^{u_1} h_2^s h_3 = A^{X_{\mathcal{B}}}$ $x_1, x_2, u, v \in_R \mathbf{Z}_q$ $B := g_1^{x_1} g_2^{x_2}$ $a := (a')^u g^v$ $b := (b' b''^{-t})^{su} A^v = A^{wu+v}$ $c := \mathcal{H}(A \vert\vert B \vert\vert z \vert\vert a \vert\vert b)$ $c' := c/u$		
	$\xrightarrow{c'}$	
		$r' := c' X_{\mathcal{B}} + w$
	$\xleftarrow{r'}$	
$r := r' u + v$ $g^r \overset{?}{=} h^c a,\ A^r \overset{?}{=} z^c b$		
$(sig(A, B) = (z, a, b, r))$		(E_1, E_2)

Figure 4.17: Withdrawal in the simplified system from [FTY98].

The proof V_1 proves that for values $x, y, z \in \mathbf{Z}_q$ known to $\mathcal{U}$,

$$E_1 = f_3^x g_2^y \quad \text{and} \quad E_2 = g_3^x 1^z;$$

[16] Otherwise, $\mathcal{U}$ would know two representations of $g_1^{u_1} g_2$ $((u_1; 1)$ and $(0; 0))$ and could double-spend without being identified.

hence, it shows that (E_1, E_2) is an ElGamal encryption based on the public key f_3.

The proof V_2 proves that for values $a, b, c, d, e, f \in \mathbf{Z}_q$ known to $\mathcal{U}$,

$$g_3 = I'^a g_1^b g_4^c \quad \text{and} \quad E_1 = g_2^a f_3^d \quad \text{and} \quad I = I'^a g_3^e g_4^f;$$

hence, assuming that $\mathcal{U}$ knows only one representation of E_1 with respect to (g_2, f_3) [17], we have $a = y$ and $d = x$ with the values x, y from the proof V_1. We can then rewrite the equations above as

$$I' = g_3^{1/y} g_1^{-b/y} g_4^{-c/y} \quad \text{and} \quad E_1 = g_2^y f_3^x \quad \text{and} \quad I' = I^{1/y} g_3^{-e/y} g_4^{-f/y};$$

with $I := g_1^{u_1}$ and by defining $y := s$ and $x := m$, we obtain

$$I' = g_3^{1/s} g_1^{-b/s} g_4^{-c/s} \quad \text{and} \quad E_1 = g_2^s f_3^m \quad \text{and} \quad I' = g_1^{u_1/s} g_3^{-e/s} g_4^{-f/s}.$$

Assuming that $\mathcal{U}$ knows only one representation of I' with respect to (g_1, g_3, g_4), we get $e = -1$, $b = -u_1$, $c = f$. By setting $t := -c/s$, we then have the construction of I' as

$$I' = g_1^{u_1 s^{-1}} g_3^{s^{-1}} g_4^t.$$

After $\mathcal{B}$ has verified V_1 and V_2, the value A is blindly signed; the signature protocol is almost identical to the one in [Bra93]. The only modification is the inclusion of the additional generator g_4 and of the value t chosen by $\mathcal{U}$ at the beginning of the withdrawal protocol.

A is blinded by use of the blinding factor s chosen before. $\mathcal{U}$ could at this point cheat by using some $s' \neq s$ and thus obtain an untraceable coin (since it no longer contains the value g_2^s which is encrypted in (E_1, E_2), but $g_2^{s'}$). However, during payment he is required to present a coin of a specific form. This forces him to use s for blinding A. We'll see this below.

$\mathcal{B}$ stores the encryption (E_1, E_2) in his withdrawal database.

Payment

In the payment protocol shown in Figure 4.18, $\mathcal{U}$ computes $A_1 := g_1^{u_1} g_2^s$. Only if the value s was chosen as the blinding factor for A, the verification

$$A_1 g_3 \stackrel{?}{=} A$$

performed by $\mathcal{S}$ will hold. $\mathcal{U}$ also provides the value A_2 and proves that $(A_1, A_2) = (g_1^{u_1} g_2^s, f_2^s)$ forms an inverted ElGamal encryption of $I = g_1^{u_1}$. The proof V_3 proves this by showing that for values $j, k, l \in \mathbf{Z}_q$ known to $\mathcal{U}$,

$$A_1 = g_2^j g_1^k \quad \text{and} \quad A_2 = f_2^j 1^l.$$

Since $A_1 = A/g_3$ by the verification performed above, and since the withdrawal protocol ensures that $A = g_1^{u_1} g_2^s g_3$, $\mathcal{S}$ is convinced that $A_1 = g_1^{u_1} g_2^s$. Assuming, as usual, that $\mathcal{U}$ knows only one representation of A_1, we have $j = s$ and $k = u_1$. The proof V_3 then shows that $A_2 = f_2^s$ and thus that (A_1, A_2) indeed is an inverted ElGamal encryption of $I = g_1^{u_1}$.

In the computation of V_3, the value A' is replaced by the value $B = g_1^{x_1} g_2^{x_2}$ created during withdrawal. This makes double-spender identification without the help of $\mathcal{T}$ possible, as will be seen below.

Double-spender Identification

If $\mathcal{U}$ spends the same coin more than once, $\mathcal{B}$ will end up with two payment transcripts containing tuples (c, r, r_1, r_2), (c', r', r_1', r_2') resulting from two proofs V_3.

It's not said whether the proof V_3 is executed interactively or not; in both cases, with high probability $c \neq c'$ (otherwise $\mathcal{S}$ is trying to double-deposit a coin). For the interactive case this is clear; in the noninteractive setting, $\mathcal{U}$ computes the challenges himself as $c := \mathcal{H}(A_1 || B || g_2 || g_1 || A_2 || B' || f_2 || 1 || \text{info})$

[17] Otherwise, $\mathcal{U}$ could compute $\log_{g_2} f_3$.

$$\begin{array}{ccc}
\mathcal{U} & & \mathcal{S} \\
(A, B, (z,a,b,r)) & & \\
\hline
A_1 := g_1^{u_1} g_2^s = A/g_3 & & \\
A_2 := f_2^s & & \\
V_3 := \mathit{Proof}_{REP+EQLOG}\big((A_1,g_2),g_1,(A_2,f_2),1\big) & & \\
& \xrightarrow[(z,a,b,r),V_3]{A_1,A_2,A,B} & \\
& & A_1 \overset{?}{\neq} 1 \\
& & A_1 g_3 \overset{?}{=} A \\
& & \text{verify } \mathit{sig}(A,B) = (z,a,b,r) \\
& & \text{verify } V_3 \\
\hline
& & (A,B,(z,a,b,r),A_1,A_2,V_3)
\end{array}$$

Figure 4.18: Payment in the simplified system from [FTY98].

and $c' := \mathcal{H}(A_1||B||g_2||g_1||A_2||B'||f_2||1||\text{info}')$. As info $\neq$ info$'$ (these values consist of data unique to each payment, e.g. date/time and $I_\mathcal{S}$), with high probability $c \neq c'$. Now, since the same values x_1, x_2 are used in both proofs V_3, $\mathcal{B}$ can compute

$$\frac{r_1 - r_1'}{c - c'} = u_1.$$

Anonymity Revocation

The anonymity can be revoked by $\mathcal{T}$ by use of his secret key $X_\mathcal{T}$; the two different kinds of anonymity revocation are performed as follows:

- Payment-based (owner tracing): $\mathcal{T}$ is given the pair (A_1, A_2) observed in a payment; $\mathcal{T}$ computes $A_1/A_2^{1/X_\mathcal{T}} = I$ and thus obtains the account number of $\mathcal{U}$.

- Withdrawal-based (coin tracing): $\mathcal{T}$ is given a pair (E_1, E_2) observed in a withdrawal; $\mathcal{T}$ computes $E_1/E_2^{X_\mathcal{T}} = g_2^s$ and then $A = I g_2^s g_3$. The coin A can be put on a blacklist for recognizing it when it is spent.

Security for $\mathcal{U}$

In [FTY98], a proof was sketched that if anonymity is compromised, then the Decision Diffie-Hellman problem can be solved. As proven in [FTY96], $\mathcal{U}$'s anonymity can only be computational.

$\mathcal{U}$ can't be falsely accused of overspending, as this would require forging a payment transcript; as for [Bra93], this is assumed to be infeasible unless one can compute discrete logarithms or knows how to represent the coin.

If $\mathcal{U}$ indeed overspends, the values u_1, s that serve to represent the coin A become known to $\mathcal{B}$ at deposit and even to $\mathcal{S}$ during payment, if $\mathcal{U}$ spends a coin twice at the same $\mathcal{S}$. Anyone knowing these values could then do further overspending.

Security for $\mathcal{B}$ and $\mathcal{S}$

The blind signature protocol used during withdrawal is as secure as [Bra93], since the modifications don't affect security. A proof sketch appeared in [FTY98]. Hence, the security for $\mathcal{B}$ and $\mathcal{S}$ is as in [Bra93].

Security for $\mathcal{T}$

For $\mathcal{T}$ being able to trace, the encryption (E_1, E_2) has to be computed correctly and the values A_1, A_2 must be of the correct structure (i.e. they form an ElGamal encryption of $\mathcal{U}$'s identity). These facts should be proven by the proofs V_1, V_2, V_3. $\mathcal{T}$ has to trust $\mathcal{B}$ and $\mathcal{S}$ that they perform the proof protocols with $\mathcal{U}$ correctly, i.e. that they don't cooperate with a cheating $\mathcal{U}$ or cheat themselves. It was shown that $\mathcal{U}$ could cheat by himself only if he is able to compute discrete logarithms; hence, under the assumption that this is infeasible, $\mathcal{T}$ will be able to trace.

4.3.8 The Off-line System from [dST98]

In [dST98], a fair off-line e-cash system based on a modification of the scheme from [Bra93] was presented. The trustee is passive and owner tracing directly provides the identity / account number of the owner of the coin (instead of a value that the larger withdrawal database has to be searched for).

The security of the scheme is based on the representation problem and on the Decision Diffie-Hellman problem.

System Setup

Two large primes p, q with $q|p-1$ are generated (it was not mentioned by whom; let's assume they are chosen by $\mathcal{B}$ in a publicly verifiable pseudorandom manner). G_q is defined as the unique subgroup of $\mathbf{Z}_p^*$ of order q. The description of G_q (i.e. p and q) is known to all parties.

$\mathcal{T}$ chooses a generator $g_\mathcal{T}$ and two secret keys $x_\mathcal{T}, y_\mathcal{T} \in \mathbf{Z}_q^*$; the corresponding public keys are

$$h_{CT} = g_\mathcal{T}^{x_\mathcal{T}^{-1}}, h_{OT} = g_\mathcal{T}^{y_\mathcal{T}^{-1}},$$

which are published together with $g_\mathcal{T}$. The indices CT, OT refer to the use of the keys (coin tracing and owner tracing).

$\mathcal{B}$ chooses a secret key $x \in_R \mathbf{Z}_q^*$ and three generators g, g_1, g_2 of G_q; it is assumed that discrete logarithms of them with respect to each other are unknown to all participants. $\mathcal{B}$ publishes these generators and the corresponding public keys $h := g^x, h_1 := g_1^x, h_2 := g_2^x, h_\mathcal{T} := g_\mathcal{T}^x$. Finally, $\mathcal{B}$ chooses a collision-resistant one-way hash function $\mathcal{H}$ that maps $\{0,1\}^*$ to $\mathbf{Z}_{2^k}$ for a security parameter k.

Account Opening

At account opening, $\mathcal{U}$ chooses a secret value $x_u \in \mathbf{Z}_q^*$ and computes his public identity or "account number" as $Id_\mathcal{U} := g_1^{x_u}$ and proves to $\mathcal{B}$ that he knows how to represent $Id_\mathcal{U}$ with respect to g_1. $\mathcal{U}$ and $\mathcal{B}$ independently compute $P := h_1^{x_u} = (Id_\mathcal{U})^x$.

Withdrawal

The withdrawal protocol consists of two phases which are shown in Figures 4.19 and 4.20.

In the first phase, $\mathcal{U}$ gives $\mathcal{B}$ the information that will allow $\mathcal{T}$ to perform coin tracing: $\mathcal{U}$ chooses a random value $s \in_R \mathbf{Z}_q$ and generates an ElGamal encryption $(g^s g_\mathcal{T}^s, h_{CT}^s)$ of the value g^s; this value will be the blinding factor used to blind the coin in the second phase. With $F := g_\mathcal{T} g$, the proof $\mathit{Proof}(ct) = \mathit{Proof}_{EQLOG}(r_a, F, G, h_{CT}, ct)$ is used to convince $\mathcal{B}$ of the correct construction of the encryption, i.e. it proves that

$$G = F^s = (g_\mathcal{T} g)^s \quad \text{and} \quad ct = h_{CT}^s.$$

$$\begin{array}{ccc}
\mathcal{U} & & \mathcal{B} \\
(F := g_{\mathcal{T}} g) & & (F := g_{\mathcal{T}} g) \\
\hline
s \in_R \mathbf{Z}_q & & \\
G := F^s = g_{\mathcal{T}}^s g^s & & \\
ct := h_{CT}^s & & \\
\mathit{Proof}(ct) := \mathit{Proof}_{EQLOG}(r_a, F, G, h_{CT}, ct) & & \\
 & \xrightarrow{ct,\, G,\, \mathit{Proof}(ct)} & \\
 & & \text{verify } \mathit{Proof}(ct) \\
\hline
(s, G) & & (ct, G)
\end{array}$$

Figure 4.19: Withdrawal in the system from [dST98], part 1.

The "authentication response" r_a is used to link the withdrawal to the prior authentication between the parties and could be $\mathcal{U}$'s last response in the authentication protocol. $\mathcal{U}$ has to prove his identity somehow to be able to withdraw coins the value of which is debited from his account; if the withdrawal is performed electronically, the authentication protocol replaces the identity check with a paper document (passport, driver's licence etc.). The value *ct* is stored by $\mathcal{B}$ for possible later coin tracing.

The second phase of the withdrawal protocol consists mainly of the execution of a restrictive blind signature protocol presented in [dST98] and similar to the one from [Bra93]. It yields a blind signature (z, c, r) by $\mathcal{B}$ on $M||coin = ot||D||E||Id_{\mathcal{U}} g_2 g_{\mathcal{T}}^s$. Both parties independently compute the blinded coin *Blindcoin*; note that $Blindcoin = coin \times g^s$. The purpose of the values *ot*, D, E grouped in the additional message $M := ot||D||E$ is to ensure the possibility of owner tracing and to make double-spender identification possible without the help of $\mathcal{T}$. We'll see how this works below.

Payment

The payment protocol is shown in Figure 4.21; $\mathcal{U}$ uses the values a, b used to compute D and E during withdrawal to create a proof $\mathit{Proof}(ot) := (c_{OT}, r_1, r_2) = \mathit{Proof}_{REP+EQLOG}(msg, g_{\mathcal{T}}, g_1, C, h_{OT}, ot)$, where $msg := Id_{\mathcal{S}}||t$ is $\mathcal{S}$' account number concatenated with the payment date/time t. This proof convinces $\mathcal{S}$ that $\mathcal{U}$ knows a representation of $C := coin/g_2$ with respect to $(g_{\mathcal{T}}, g_1)$ and that the exponent of $g_{\mathcal{T}}$ in it is equal to $\log_{h_{OT}} ot$; in other words, for values $x, y \in \mathbf{Z}_q$ known to $\mathcal{U}$,

$$C = g_{\mathcal{T}}^x g_1^y \quad \text{and} \quad ot = h_{OT}^x.$$

Assuming that $\mathcal{U}$ knows only one such representation [18] of C and that the blind signature protocol used during withdrawal is restrictive, we have $x = s$ and $y = x_{\mathcal{U}}$. Hence, (C, ot) is an ElGamal encryption of $\mathcal{U}$'s identity based on $\mathcal{T}$'s public key h_{OT}.

$\mathcal{U}$ sends $(M$, *coin*, $sig(M||coin) = (z, c, r)$, $\mathit{Proof}(ot), C, ot)$ to $\mathcal{S}$ who accepts the payment if the verification of $sig(M||coin)$ and $\mathit{Proof}(ot)$ is successful. If all participants are honest, the verification equation for $sig(M||coin)$

$$c \overset{?}{=} \mathcal{H}(M||coin||z||g^r h^c||coin^r z^c)$$

holds since

$$g^r h^c = g^{ur_0+v} h^c = g^{u(\omega - c_0 x)} g^v h^c = g^{u\omega} g^{cx} g^v h^c = A_0^u g^v = A$$

and

$$coin^r z^c = coin^{ur_0+v} z^c = coin^{u(\omega - c_0 x)} coin^v z^c = coin^{u\omega} coin^{-cx} coin^v z^c$$

[18] Otherwise, $\mathcal{U}$ could compute $\log_{g_1} g_{\mathcal{T}}$.

$$
\begin{array}{ccc}
\mathcal{U} & & \mathcal{B} \\
(s, G = g_{\mathcal{T}}^{s} g^{s}) & & (G = g_{\mathcal{T}}^{s} g^{s}) \\
\hline
\mathit{Blindcoin} := \mathit{Id}_{\mathcal{U}} g_2 G & & \mathit{Blindcoin} := \mathit{Id}_{\mathcal{U}} g_2 G \\
 & & \omega \in_R \mathbf{Z}_q^{*} \\
 & & z_0 := \mathit{Blindcoin}^{x} = (\mathit{Id}_{\mathcal{U}} g_2 G)^{x} \\
 & & A_0 := g^{\omega} \\
 & & B_0 := \mathit{Blindcoin}^{\omega} \\
 & \xleftarrow{\quad z_0, A_0, B_0 \quad} & \\
u, v, a, b \in_R \mathbf{Z}_q^{*} & & \\
A := A_0^{u} g^{v} & & \\
B := B_0^{u} \mathit{Blindcoin}^{v} / A^{s} & & \\
z := z_0 / h^{s} & & \\
\mathit{ot} := h_{\mathit{OT}}^{s} & & \\
D := g_1^{a} g_{\mathcal{T}}^{b} & & \\
E := h_{\mathit{OT}}^{b} & & \\
M := \mathit{ot} || D || E & & \\
\mathit{coin} := \mathit{Id}_{\mathcal{U}} g_2 g_{\mathcal{T}}^{s} & & \\
c := \mathcal{H}(M || \mathit{coin} || z || A || B) & & \\
c_0 := c / u & & \\
 & \xrightarrow{\qquad c_0 \qquad} & \\
 & & r_0 := \omega - c_0 x \\
 & \xleftarrow{\qquad r_0 \qquad} & \\
A_0 \stackrel{?}{=} g^{r_0} h^{c_0} & & \\
B_0 \stackrel{?}{=} \mathit{Blindcoin}^{r_0} z_0^{c_0} & & \\
r := u r_0 + v & & \\
\hline
(\mathit{coin}, (z, c, r), a, b, \mathit{ot}) & &
\end{array}
$$

Figure 4.20: Withdrawal in the system from [dST98], part 2.

$\mathcal{U}$ $\qquad\qquad\qquad\qquad$ $\mathcal{S}$

$(s, \mathit{coin}, (z, c, r), a, b, \mathit{ot})$

$$\mathit{msg} := \mathit{Id}_{\mathcal{S}}||t$$
$$C := \mathit{coin}/g_2$$
$$D := g_1^a g_{\mathcal{T}}^b$$
$$E := h_{OT}^b$$
$$c_{OT} := \mathcal{H}(\mathit{msg}||g_{\mathcal{T}}||g_1||C||h_{OT}||\mathit{ot}||D||E)$$
$$r_1 := b - c_{OT}s, r_2 := a - c_{OT}x_u$$
$$\mathit{Proof}(\mathit{ot}) := (c_{OT}, r_1, r_2)$$
$$= \mathit{Proof}_{REP+EQLOG}(\mathit{msg}, g_{\mathcal{T}}, g_1, C, h_{OT}, \mathit{ot})$$

$$\xrightarrow[\mathit{Proof}(\mathit{ot}), C, \mathit{ot}]{M, \mathit{coin}, (z, c, r)}$$

verify $\mathit{sig}(M||\mathit{coin}) = (z, c, r)$

verify $\mathit{Proof}(\mathit{ot})$

$(M, \mathit{coin}, (z, c, r), \mathit{Proof}(\mathit{ot}), C, \mathit{ot})$

Figure 4.21: Payment in the system from [dST98].

$$= \left(\frac{\mathit{Blindcoin}}{g^s}\right)^{\omega u}\left(\frac{\mathit{Blindcoin}}{g^s}\right)^{v} = \left(\frac{B_0^u}{A_0^{su}}\right)\left(\frac{\mathit{Blindcoin}^v}{g^{sv}}\right) = \frac{B_0^u \mathit{Blindcoin}^v}{(A_0^u g^v)^s} = \frac{B_0^u \mathit{Blindcoin}^v}{A^s} = B,$$

and therefore

$$\mathcal{H}(M||\mathit{coin}||z||g^r h^c||\mathit{coin}^r z^c) = \mathcal{H}(M||\mathit{coin}||z||A||B) =: c.$$

The verification equation for $\mathit{Proof}(\mathit{ot})$

$$c_{OT} \overset{?}{=} \mathcal{H}(\mathit{msg}||g_{\mathcal{T}}||g_1||C||h_{OT}||\mathit{ot}||g_{\mathcal{T}}^{r_1} g_1^{r_2} C^{c_{OT}}||h_{OT}^{r_1}\mathit{ot}^{c_{OT}}) \overset{?}{=} \mathcal{H}(\mathit{msg}||g_{\mathcal{T}}||g_1||C||h_{OT}||M). \tag{4.9}$$

holds since

$$g_{\mathcal{T}}^{r_1} g_1^{r_2} C^{c_{OT}} = g_{\mathcal{T}}^{b - c_{OT}s} g_1^{a - c_{OT}x_u} C^{c_{OT}} = g_{\mathcal{T}}^b g_1^a g_{\mathcal{T}}^{-c_{OT}s} g_1^{-c_{OT}x_u} C^{c_{OT}} = g_{\mathcal{T}}^b g_1^a = D$$

and

$$h_{OT}^{r_1}\mathit{ot}^{c_{OT}} = h_{OT}^{b - c_{OT}s}\mathit{ot}^{c_{OT}} = h_{OT}^b = E.$$

Notice that $\mathcal{S}$ has to verify both equalities of Equation 4.9; the first one holds if $\mathcal{U}$ knows the representations as required, while the second equality holds if $M = \mathit{ot}||D||E$. If the second equality was not checked, $\mathcal{U}$ could compute c with arbitrary values $D' := g_1^{a'} g_{\mathcal{T}}^{b'} \neq D$, $E' := h_{OT}^{b'} \neq E$ and thus prevent tracing after double-spending.

Double-spender Identification

If $\mathcal{U}$ spends the same coin more than once, $\mathcal{B}$ will end up with values

$$c_{OT} = \mathcal{H}(\mathit{msg}||g_{\mathcal{T}}||g_1||C||h_{OT}||\mathit{ot}||D||E),\ r_2 = a - c_{OT}x_{\mathcal{U}}$$
$$c'_{OT} = \mathcal{H}(\mathit{msg}'||g_{\mathcal{T}}||g_1||C||h_{OT}||\mathit{ot}||D||E),\ r'_2 = a - c'_{OT}x_{\mathcal{U}},$$

where msg and msg' will differ (otherwise $\mathcal{S}$ is trying to double-deposit a coin). Thus, with high probability we have $c_{OT} \neq c'_{OT}$, and $\mathcal{B}$ can compute

$$\frac{r'_2 - r_2}{c_{OT} - c'_{OT}} = x_{\mathcal{U}}.$$

Anonymity Revocation

The anonymity can be revoked by $\mathcal{T}$ by use of his secret keys $x_\mathcal{T}, y_\mathcal{T}$; the two different kinds of anonymity revocation are performed as follows:

- Payment-based (owner tracing): $\mathcal{T}$ is given the values *coin* and *ot* observed in a payment; $\mathcal{T}$ computes $(coin/g_2)/(ot)^{y_\mathcal{T}} = Id_\mathcal{U}$ and thus obtains the account number of $\mathcal{U}$.
- Withdrawal-based (coin tracing): $\mathcal{T}$ is given a *ct* observed in a withdrawal and the withdrawing $\mathcal{U}$'s identity $Id_\mathcal{U}$; $\mathcal{T}$ computes $(ct)^{x_\mathcal{T}} = g_\mathcal{T}^s$ and then $coin = Id_\mathcal{U} g_2 g_\mathcal{T}^s$. The coin *coin* can be put on a blacklist for recognizing it when it is spent.

Security for $\mathcal{U}$

$\mathcal{U}$'s anonymity is preserved due to the blindness of the restrictive signature protocol executed during withdrawal. It is based on the protocol from [Bra93], which is unconditionally blind. The additional values $G := F^s$, $ct := h_{CT}^s$, and $ot := h_{OT}^s$ reduce the blindness to computational. Linking withdrawals to payments by use of these values (i.e. linking values G, ct from a withdrawal to the corresponding *ot* from payment) without knowing $\log_{g_\mathcal{T}} h_{OT}$ or $\log_{g_\mathcal{T}} h_{CT}$ is infeasible under the assumption that the Decision Diffie-Hellman problem is hard to solve. An algorithm that for values h_{CT}, $ct = h_{CT}^s$, h_{OT}, *ot* could decide whether $ot = h_{OT}^s$ could be used to solve the Decision Diffie-Hellman problem

$$\text{given } g, g^a, g^b, x, \text{ decide } x \overset{?}{=} g^{ab}$$

by setting $h_{CT} := g$, $ct := g^a$, $h_{OT} := g^b$, $ot := x$.

It is assumed that the proofs *Proof*(*ct*) and *Proof*(*ot*) leak no information that could compromise $\mathcal{U}$'s anonymity.

$\mathcal{U}$ can't be falsely accused of overspending, as this would require forging a payment transcript; as for [Bra93], this is assumed to be infeasible unless one can compute discrete logarithms or knows how to represent the coin.

If $\mathcal{U}$ indeed overspends, the values $x_\mathcal{U}, s$ that serve to represent the coin become known to $\mathcal{B}$ at deposit and even to $\mathcal{S}$ during payment, if $\mathcal{U}$ spends a coin twice at the same $\mathcal{S}$. Anyone knowing these values could then do further overspending.

Security for $\mathcal{B}$

The blind signature protocol used during withdrawal is assumed to be as secure as the one from [Bra93], since $\mathcal{B}$ doesn't give any additional information and thus the modifications don't affect $\mathcal{B}$'s security.

Security for $\mathcal{S}$

There is no way to prevent false accusations of double-depositing. However, this is not a serious problem ($\mathcal{B}$ wouldn't usually make such false accusations, as there is no money to be gained) and can be solved easily (by including a signature by $\mathcal{S}$ in every deposit request). Theft or extortion of coins paid to $\mathcal{S}$ is prevented by including $Id_\mathcal{S}$ in the payment transcript and thus making the coin depositable only by $\mathcal{S}$.

Security for $\mathcal{T}$

For $\mathcal{T}$ being able to trace, the encryptions *ct* and *ot* have to be computed correctly. These facts should be proven by the proofs *Proof*(*ct*), *Proof*(*ot*). $\mathcal{T}$ has to trust $\mathcal{B}$ and $\mathcal{S}$ that they perform the proof protocols with $\mathcal{U}$ correctly, i.e. that they don't cooperate with a cheating $\mathcal{U}$ or cheat themselves. It is assumed that $\mathcal{U}$ can't cheat during the execution of the proofs.

Chapter 5

Self-Escrowing

In [PS00], self-escrow-based e-cash against user blackmailing was proposed. Instead of a trustee it is the blackmailed user himself who helps to trace his coins, i.e. coins withdrawn by him or by the blackmailer. Thus, the risk of misuse of the revocation ability is eliminated. In this chapter, the results of [PS00] are briefly presented; then the possibility to generalize the approach used there is investigated.

5.1 Previous Results

There are various scenarios in which anonymity revocation might occur under circumstances the users wouldn't approve. Such scenarios are, e.g., a not-so-trustworthy trustee, an "Orwellian" future government, or simply some attacker gaining access to the trustee's secrets.

In [PS00], a self-escrow-based e-cash system was presented which provides coin tracing, i.e. which is secure against user blackmailing. This specific type of crime was addressed because of its peculiar characteristics in the digital world. As pointed out in the article, user blackmailing is much easier in an e-cash system than with conventional cash, as in most cases the money delivery does not require any physical contact between the user and the blackmailer [1]. In addition, in an anonymous e-cash system there is no way to make the extorted coins recognizable as can be done with paper-based cash by simply registering the serial numbers of the banknotes.

Owner tracing can't be implemented by means of self-escrowing, as the user would have to trace himself. As will be seen below, this should be kept in mind when transforming trustee escrow-based systems (with owner tracing) into self-escrow-based ones.

5.1.1 Replacing the Trustee

The basic idea from [PS00] is taking an escrow-based e-cash system and letting the user play the role of the trustee. This is viable only if certain requirements are met by the system:

Minimum trust (R1): The trustee mustn't need to be trusted by the bank or the recipients for anything else but tracing.

Passivity (R2): The trustee must be passive (at least) in withdrawals, as otherwise the blackmailer could force the user to not only withdraw the money, but also to play his trustee role wrongly.

Minimum involvement (R3): The payment and deposit protocols must work without the bank or the recipients having to input any information about the trustee (e.g. a public key), as such information would allow them to link the payments to the user.

[1] The blackmailer might encounter the user once at the beginning, but usually this is without risk to him, since $\mathcal{U}$ hasn't had time to inform law enforcement authorities.

The requirement R2 is a bit too restrictive and should be loosened by adding the following:

> Alternatively, the bank has to verify that the trustee plays his role correctly.

Most escrow-based e-cash systems don't fulfill all of the requirements above. Typically, the public key of the trustee is needed at payment or deposit time or both, hence R3 is violated. However, this is often not the case anymore after owner tracing has been removed from the scheme.

In the next section, we'll see why the fulfillment of the three requirements above guarantees that the security of the system is not diminished.

5.1.2 Self-Escrowing in the System from [FTY96]

In [PS00], a concrete instantiation of self-escrowing was implemented based on the system from [FTY96]. The modifications to the system were the following:

- At account opening, $\mathcal{U}$ additionally gives $\mathcal{B}$ a public key pk_{trace} and proves knowledge of the corresponding secret key sk_{trace}; $\mathcal{U}$ signs pk_{trace} to prevent dispute about it,
- pk_{trace} is used (instead of $\mathcal{T}$'s public key) at withdrawal to encrypt A_2 to enc.

In case $\mathcal{U}$ is blackmailed and pays the blackmailer, $\mathcal{B}$ can retrieve the value enc related to the withdrawal of the extorted money. $\mathcal{U}$ can then decrypt the tracing information with sk_{trace}; as can be seen, it is crucial that $\mathcal{U}$ actually is in possession of sk_{trace}. This secret key is created and stored during account opening; ideally, $\mathcal{U}$ may have an arbitrary number of backups of it; like this, as long as the blackmailer can't be sure of having found and destroyed all backups, he risks being traced. The blackmailer can force $\mathcal{U}$ to open a new account; since $\mathcal{U}$ is to some extent under the blackmailer's control during the attack, it must be guaranteed that during account opening, $\mathcal{U}$ has a possibility to make a backup of sk_{trace}. This must happen in a way unobservable to the blackmailer, i.e. we should require that $\mathcal{U}$ opens the account alone in a room shielded against transmissions. Furthermore, it must be ensured that $\mathcal{U}$ can freely choose sk_{trace}; otherwise, the blackmailer could choose it and give $\mathcal{U}$ an electronic device which contains sk_{trace} and permits account opening, but doesn't reveal sk_{trace} to $\mathcal{U}$, thus preventing a backup. As can be seen, all this is no trivial issue; in a concrete implementation, the protective measures would be chosen according to their cost and to the risks they protect against. We'll not go into further detail here.

The only actual modification to the original scheme from [FTY96] is the replacement of $\mathcal{T}$'s public key with that of $\mathcal{U}$, i.e. $\mathcal{U}$ plays the role of $\mathcal{T}$. One might at first sight object that the original scheme doesn't fulfill all of the requirements claimed to be necessary for being able to simply replace $\mathcal{T}$ like this: indeed it is true that $\mathcal{T}$'s public key is needed at payment, when $\mathcal{U}$'s identity is encrypted with it, in violation of R3. But this is done to allow for owner tracing, which is incompatible with self-escrowing. Thus, the parts of the scheme from [FTY96] needed for owner tracing were dropped altogether.

5.2 Further Discussion

In this section, the transformation of existing escrow-based e-cash schemes in ones with self-escrowing is investigated. Since trustee replacement, as described above, is the key concept, its security is investigated first of all. Afterwards, self-escrowing is implemented in the system from [CMS96], in the modular system from [FTY98], and in the scheme from [dST98]. After these concrete instantiations, we'll try to make some general statements about implementing self-escrowing in escrow-based systems with trustee.

5.2.1 Security Issues

In the following, we'll check that when the requirements from Subsection 5.1.1 are met, the security for all parties isn't diminished.

Security for $\mathcal{U}$

When $\mathcal{U}$ takes over $\mathcal{T}$'s role, he gains control over the anonymity revocation (namely coin tracing) in the system and thus needn't trust any third party $\mathcal{T}$ anymore; this is a security gain. On the other hand, $\mathcal{U}$ himself becomes responsible for assuring the possibility for anonymity revocation and might prevent it by not playing his trustee role correctly (on purpose or by mistake). But requirement R2 guarantees that after trustee replacement, it is still as hard as before to obtain untraceable coins.

Requirement R3 ensures that $\mathcal{U}$'s anonymity isn't compromised, as it means that in the self-escrow-based system, the bank or the recipients don't have to input any information about the "new trustee" $\mathcal{U}$ during payment or deposit. Therefore, $\mathcal{B}$ or $\mathcal{S}$ or both together can't link payments to withdrawals, and $\mathcal{U}$'s anonymity is preserved.

Security for $\mathcal{B}$

The first requirement of not having to trust $\mathcal{T}$ for anything else but tracing ensures that the collusion of $\mathcal{T}$ with $\mathcal{U}$ (or anyone else) doesn't compromise the security of the system. Hence, in a self-escrow-based system fulfilling this requirement coins are just as hard to forge as in the underlying trustee escrow-based system.

When it comes to double-spending, it is clear that in the underlying trustee escrow-based system, the identification of double-spenders must be possible without $\mathcal{T}$'s help. Thus, implicitly "tracing" in the first requirement means coin tracing, not double-spender tracing.

Security for $\mathcal{S}$

As for $\mathcal{B}$, the first requirement guarantees that the implementation of self-escrowing by trustee replacement doesn't compromise $\mathcal{S}$' security in the system. If in the self-escrow-based system an attack against $\mathcal{S}$ can be performed, this is not due to the trustee replacement, but to a weakness of the underlying system.

5.2.2 Self-Escrowing in the System from [CMS96]

The system from [CMS96] fulfills the requirements R1, R2, R3 from Subsection 5.1.1: $\mathcal{T}$ being passive, he doesn't participate in the regular protocols (withdrawal and payment), but performs only anonymity revocation; $\mathcal{T}$ is therefore trusted only for tracing (R1). During withdrawal, $\mathcal{T}$ is passive; $\mathcal{U}$ encrypts the tracing information d and proves its correct construction to $\mathcal{B}$ (R2). The public key of $\mathcal{T}$ isn't needed during payment or deposit (R3).

In the following, the modifications to the original scheme which implement self-escrowing are described:

At account opening, $\mathcal{U}$ provides an additional public key $pk_{trace} := g_2^{sk_{trace}}$ and proves knowledge of the corresponding secret key sk_{trace}. $\mathcal{U}$ signs pk_{trace} with his regular signature key to prevent dispute about it.

At withdrawal, the encrypted tracing information $d := y_{\mathcal{T}}^{\alpha}$ is replaced by $d := pk_{trace}^{\alpha}$. Consequently, the proof U about the correct construction of h_w and d becomes

$$U := PLOGEQ(\epsilon, g_1, (h_w/g_2), d, pk_{trace}).$$

Since $log_{g_1}(h_w/g_2) = log_d(pk_{trace})$, $\mathcal{B}$'s verification of U will succeed.

At payment (both in the on-line and the off-line system), pk_{trace} is never handed to $\mathcal{S}$. Neither is any other value that would make it possible for $\mathcal{S}$ or $\mathcal{B}$ or both to link the payment to the withdrawal of the coin. Namely,

- in the on-line system, $\mathcal{U}$ gives $\mathcal{S}$
  - $c\#, h_p, z_p, W$, which $\mathcal{B}$ hasn't seen and which $\mathcal{B}$ can't link to any withdrawal

  - $V := \mathit{Proof}_{KLOG}(g_2, h_p/g_1) = PKLOG(\epsilon, g_2, h_p/g_1)$, which is a blind proof of knowledge [2]

- in the off-line system, $\mathcal{U}$ gives $\mathcal{S}$
  - h_p, z_p, W, t_p, which $\mathcal{B}$ hasn't seen and which $\mathcal{B}$ can't link to any withdrawal
  - $s_p := r_p - c_p\alpha$, which is blinded by α.

Tracing is done as in the original scheme: $\mathcal{B}$ retrieves the encrypted tracing information $d := pk_{trace}^{\alpha} = (g_2^{sk_{trace}})^{\alpha}$. $\mathcal{U}$ then computes $g_1 d^{1/sk_{trace}} = g_1 g_2^{\alpha} = h_p$.

5.2.3 Self-Escrowing in the Modular System from [FTY98]

As with the system from [FTY96], the requirements R1, R2, R3 are met and hence self-escrowing can be implemented by substituting $\mathcal{U}$ for $\mathcal{T}$.

At account opening, $\mathcal{U}$ provides an additional public key $pk_{trace} := g_1^{sk_{trace}}$ and proves knowledge of the corresponding secret key sk_{trace}. $\mathcal{U}$ signs pk_{trace} with his regular signature key to prevent dispute about it.

At withdrawal, the encryption (E_1, E_2) of the tracing information $(Ig_2g_4^t)^s$ is computed as

$$E_1 := (Ig_2g_4^t)^s pk_{trace}^m \quad \text{and} \quad E_2 := g_1^m,$$

thus replacing $\mathcal{T}$'s public key f_1 by pk_{trace}. As pk_{trace} isn't needed at payment or deposit time, the paid coin can't be linked to the withdrawal.

Coin tracing is performed by decrypting (E_1, E_2) with sk_{trace} to $E_1/E_2^{sk_{trace}} = (Ig_2g_4^t)^s$.

5.2.4 Self-Escrowing in the System from [dST98]

This system also fulfills the requirements for the implementation of self-escrowing by substitution of $\mathcal{U}$ for $\mathcal{T}$. However, removing the parts needed for owner tracing is a bit more complicated than in the systems presented so far, as these parts are intertwined with the mechanism that allows $\mathcal{B}$ to identify double-spenders without the help of $\mathcal{T}$.

The first step is to redefine the generator $g_{\mathcal{T}}$ (chosen by $\mathcal{T}$ in the original system) as an additional generator chosen and published by $\mathcal{B}$. If each user was to choose a "personal generator" $g_{\mathcal{U}}$ corresponding to $g_{\mathcal{T}}$, user anonymity would clearly be compromised.

$\mathcal{U}$ chooses an additional secret key sk_{trace} with related public key $pk_{trace} := g_{\mathcal{T}}^{sk_{trace}^{-1}}$ and proves knowledge of sk_{trace}; the key pair (sk_{trace}, pk_{trace}) replaces $(x_{\mathcal{T}}, h_{CT})$. The key pair $(y_{\mathcal{T}}, h_{OT})$ and therefore the computation of ot and E are dropped, as they serve the purpose of owner tracing. Hence, the message $M := ot||D||E$ now becomes $M := D$. It will be useful in identifying double-spenders.

At withdrawal, the encrypted tracing information $ct := h_{CT}^s$ is replaced by $ct := pk_{trace}^s$. Consequently, the proof $\mathit{Proof}(ct)$ about the correct construction of ct becomes

$$\mathit{Proof}(ct) := \mathit{Proof}_{EQLOG}(r_a, F, G, pk_{trace}, ct).$$

As it is true that $\log_F G = log_{pk_{trace}} ct$, $\mathcal{B}$'s verification of $\mathit{Proof}(ct)$ will succeed.

The new payment protocol is presented in Figure 5.1. At payment, $\mathcal{U}$ no longer proves knowledge of representation and equality of logarithms for the values C, ot (ot served for owner tracing and was thus dropped). Instead, he uses the proof $\mathit{Proof}(C)$ to show that he knows how to represent C with respect to $(g_{\mathcal{T}}, g_1)$.

$\mathcal{U}$ sends $\mathcal{S}$ the values M, $coin$, $sig(M||coin) = (z, c, r)$, $\mathit{Proof}(C)$, and C. The verification of $sig(M||coin)$ is as in the system with trustee, while $\mathit{Proof}(C)$ is verified by checking

$$c_C \overset{?}{=} \mathcal{H}(msg||g_{\mathcal{T}}||g_1||C||g_{\mathcal{T}}^{r_1} g_1^{r_2} C^{c_C}) \overset{?}{=} \mathcal{H}(msg||g_{\mathcal{T}}||g_1||C||M).$$

[2] $\mathcal{S}'$ view of V can be simulated in an indistinguishable way.

$$\begin{array}{ccc}
\mathcal{U} & & \mathcal{S} \\
(s, \mathit{coin}, (z, c, r), a, b) & & \\
\hline
\mathit{msg} := \mathit{Id}_{\mathcal{S}} || t & & \\
C := \mathit{coin}/g_2 & & \\
D := g_1^a g_{\mathcal{T}}^b & & \\
c_C := \mathcal{H}(\mathit{msg}||g_{\mathcal{T}}||g_1||C||D) & & \\
r_1 := b - c_C s, r_2 := a - c_C x_u & & \\
\mathit{Proof}(C) := (c_C, r_1, r_2) & & \\
= \mathit{Proof}_{REP}(\mathit{msg}, g_{\mathcal{T}}, g_1, C) & & \\
 & \xrightarrow[\mathit{Proof}(C)]{M, \mathit{coin}, \mathit{sig}(M||\mathit{coin})} & \\
 & & \text{verify } \mathit{sig}(M||\mathit{coin}) \\
 & & \text{and } \mathit{Proof}(C) \\
\hline
 & & (M, \mathit{coin}, (z, c, r), \mathit{Proof}(C), C)
\end{array}$$

Figure 5.1: Payment in a self-escrow-based system based on [dST98].

If $\mathcal{U}$ is honest, this equation holds since

$$g_{\mathcal{T}}^{r_1} g_1^{r_2} C^c = g_{\mathcal{T}}^{b-c_C s} g_1^{a-c_C x_u} C^{c_C} = g_{\mathcal{T}}^b g_1^a g_{\mathcal{T}}^{-c_C s} g_1^{-c_C x_u} C^{c_C} = g_{\mathcal{T}}^b g_1^a = D =: M.$$

As in the system with trustee, both equalities have to be verified; the first one ensures that $\mathcal{U}$ knows the representation, while the second one ensures the possibility to identify double-spenders. If it wasn't checked, $\mathcal{U}$ could replace D by $D' := g_1^{a'} g_{\mathcal{T}}^{b'} \neq D$ and thus prevent tracing after double-spending.

As h_{CT} isn't needed at payment or deposit time in the base system, so is not pk_{trace} in the self-escrow-based scheme. Thus, the paid coin (*coin*) can't be linked to the withdrawal. Coin tracing is performed by computing $(ct)^{sk_{trace}} = g_{\mathcal{T}}^s$ and then $\mathit{coin} = \mathit{Id}_{\mathcal{U}} g_2 g_{\mathcal{T}}^s$.

Double-spender Identification

Double-spender identification works just as in the scheme with trustee, although we dropped the values *ot*, E. The value E was needed only as a "buffer value" to complete the proof of equality of logarithms in the base scheme.

If $\mathcal{U}$ spends the same coin more than once, $\mathcal{B}$ will end up with values

$$c_C = \mathcal{H}(\mathit{msg}||g_{\mathcal{T}}||g_1||C||D),\ r_2 = a - c_C x_{\mathcal{U}}$$

$$c'_C = \mathcal{H}(\mathit{msg}||g_{\mathcal{T}}||g_1||C||D),\ r'_2 = a - c'_C x_{\mathcal{U}},$$

where *msg* and *msg'* will differ (otherwise $\mathcal{S}$ is trying to double-deposit a coin; recall that these values include $\mathcal{S}$' identity and data unique to a single payment). Thus, with high probability we have $c_C \neq c'_C$, and $\mathcal{B}$ can compute

$$\frac{r'_2 - r_2}{c_C - c'_C} = x_{\mathcal{U}}.$$

5.2.5 Generalization of Self-Escrowing

When an escrow-based e-cash system fulfills all of the requirements from Subsection 5.1.1, self-escrowing can be implemented by letting $\mathcal{U}$ play the trustee's role. As pointed out in Subsection 5.2.2, this is the case with the system from [CMS96]. For the systems from [FTY96], [FTY98], and [dST98] the requirements are fulfilled after one has removed all parts needed for owner tracing from the scheme.

Is there a universally valid method to transform a trustee escrow-based e-cash scheme into a self-escrow-based system? First, one should remember that self-escrowing provides only coin tracing; in the scheme with trustee, for coin tracing to be possible, some tracing information (a "coin identifier") has to be escrowed, i.e. deposited at $\mathcal{T}$. This has to take place before payment and therefore at withdrawal time (before withdrawal, the coin doesn't exist and there can't be any identifier for it; escrowing after payment would be too late).

The answer to the question above is given in the following for a few different types of escrow-based systems.

Systems with Passive Trustee

In a system with a passive trustee, $\mathcal{T}$ is trusted only for tracing: if $\mathcal{T}$ was trusted for anything else, he would have to participate also in some other than the tracing protocol. But this is not possible, as $\mathcal{T}$ is passive. (Formally speaking, we would first define the notion of "passivity" of a Turing machine $\mathcal{T}$, e.g. by saying that in all protocols with exception of those for tracing, the Turing machines representing the other participants do not interact with machine $\mathcal{T}$. Then, we might define "trust" and find that the "need to trust for more than tracing" contradicts "passivity".)

Being passive, $\mathcal{T}$ "participates" in the withdrawal only by presence of his public encryption key. $\mathcal{U}$ uses it to encrypt the tracing information; it is not used at payment or deposit time. If in a given system this requirement is not met, i.e. if $\mathcal{T}$'s public key is used at payment or deposit time, then this use of the key does not serve the purpose of coin tracing and may therefore with high probability be dropped to fulfill the requirement. The reason for this is that after the accomplishment of the withdrawal and before initiation of any payment, coin tracing, i.e. the unblinding of the blind coins $\mathcal{B}$ has seen during withdrawal, must already be possible. Therefore, any actions needed to enable coin tracing have to be taken already at withdrawal time.

Thus, all the requirements from Subsection 5.1.1 are met. Hence, in any system with a passive trustee ("passive" implies "trusted only for tracing"), self-escrowing should be possible by substituting $\mathcal{U}$ for $\mathcal{T}$.

Systems with On-line Trustee

In a system with an on-line trustee, $\mathcal{T}$ participates in the withdrawal by acting as an intermediary between $\mathcal{U}$ and $\mathcal{B}$: it is $\mathcal{T}$ who performs the blinding. Thus, $\mathcal{T}$ sees the coin in its unblinded form and can trace it. If $\mathcal{T}$ needs to be trusted only for tracing in the system, self-escrowing can be implemented by letting $\mathcal{U}$ play $\mathcal{T}$'s role. However, it has to be ensured that $\mathcal{U}$ has the possibility to store the tracing information (i.e. the blinding factors or the unblinded coins themselves) in a safe place and in a way unobservable to the blackmailer. Otherwise, after the blackmailing $\mathcal{U}$ won't be in possession of the tracing information and hence tracing won't be possible.

If $\mathcal{T}$ needs to be trusted for the correctness of blinding during withdrawal [3], implementing self-escrowing is not possible; trusted parties can only be replaced if the new party can also be trusted. $\mathcal{U}$ can't be trusted to perform the blinding in a correct manner. Letting $\mathcal{U}$ prove the correctness to $\mathcal{B}$ gives us the equivalent of a system with passive trustee, which is self-escrowable.

Pseudonym-based Systems

In a system in which $\mathcal{T}$ certifies pseudonyms for $\mathcal{U}$ as described in Section 4.2, self-escrowing can't be implemented by substituting $\mathcal{U}$ for $\mathcal{T}$, since $\mathcal{T}$ has to be trusted for more than just tracing correctly. Indeed, $\mathcal{T}$ certifies that the identity of the holder of a pseudonym is known to him. With $\mathcal{T}$ replaced by $\mathcal{U}$, anyone could open completely anonymous bank accounts; this is probably not wanted.

[3] A case where $\mathcal{T}$ needs to be trusted for the correctness of blinding would be if the ability to identify double-spenders relies on the correct behavior of $\mathcal{T}$.

To see why self-escrowing can't work in pseudonym-based systems one has to realize that the pseudonyms have to be certified by someone trusted by both $\mathcal{U}$ and $\mathcal{B}$ and therefore none of the two, but some third party: $\mathcal{T}$.

Efficiency Considerations

It is interesting to notice that after having implemented self-escrowing by the modular method of substituting $\mathcal{U}$ for $\mathcal{T}$, systems resulting from schemes with passive trustee seem (at first sight) to be much less efficient than those resulting from schemes with on-line trustee. In the former, at each withdrawal, $\mathcal{U}$ encrypts the tracing information with his own public key and proves to $\mathcal{B}$ that the encryption is correct. If the blackmailer has no control over $\mathcal{U}$ regarding the computation of this encryption, this proof of correctness might be omitted, saving computation time (but losing security at $\mathcal{U}$'s risk). However, each withdrawal would still require the computation of the encryption itself. In a system resulting from the transformation of a scheme with on-line trustee, $\mathcal{U}$ has only to store the blinding factors. Self-escrow-based systems of this kind thus seem more efficient. But if anonymity of honest users is to be preserved, the blinding factors have to be encrypted if stored within the bank. Storing them anywhere else would be difficult in a blackmailing situation, and therefore no option.

Thus, in the end, $\mathcal{U}$ has no choice but to encrypt "some data" to be stored by $\mathcal{B}$; whether this should be the blinding factors or some coin identifier depends on the size of this data in the underlying system. For maximum security one would therefore choose a system based on a passive trustee scheme and keep the proof of correctness of the encryption during withdrawal. This results in more computation, but a "strong" blackmailer couldn't anymore convince $\mathcal{U}$ to compute the encryption in a wrong manner.

Chapter 6

Strengthened Security

As we have seen, security is no trivial issue in systems as complex as off-line anonymous e-cash schemes; having dealt with user anonymity and trustee elimination in the previous chapter, we'll concentrate on security against coin forgery here; using the systems and techniques of the previous chapters and results of previous work [PS96a, PS96b, Poi98, CMS96], we'll try to build a payment system which is provably secure in the random oracle model. As stated in Subsection 2.5.2, this model allows us to obtain security proofs under the assumption that parts of the system (usually the hash functions) act like random oracles, i.e. their outputs are seen as truly random for each new query. Recall that this assumption is made in order to keep the systems efficient, i.e. there are systems provably secure even without this assumption, but their efficiency is poor. If a scheme's security can be proven in the random oracle model, then it will be secure in the "real world" under the assumption that the parts of the system modeled as random oracles have no weakness. This is much better than just being able to say "we can't prove it, but nobody has broken it so far".

The unforgeability of coins relies on the security of the signature scheme used for issuing coins; in [PS96a], security proofs for nonblind signature schemes in the random oracle model were addressed. Since, however, anonymity for $\mathcal{U}$ is implemented by use of blind signatures, it is desirable to have security proofs for blind signature schemes. These were presented in [PS96b], but they hold only if the number of issued signatures (i.e., of withdrawn coins) is poly-logarithmically bounded [1]. In [PS97], defined by the authors themselves as "the full version" of [PS96a] and [PS96b], the results from the two articles were made more precise, but we won't need this degree of precision here.

In [Poi98], Pointcheval presented a blind signature scheme which is provably secure even for polynomially many signatures. The key concepts from [PS96a, PS96b, Poi98] are explained in this chapter; then, we proceed to construct a complete on-line escrow-based payment system with passive trustee which is provably secure against coin forgery in the random oracle model (for polynomially many withdrawn coins). We'll see whether it is possible to implement self-escrowing in the scheme, and whether payments can be made off-line.

6.1 Nonblind Signatures

6.1.1 Types of Signature Schemes

In [PS96a], only signature schemes of the following kind were considered: the signature on a message m is a triple (σ_1, h, σ_2) such that for a hash function f with k output bits for a security parameter k,

$$h := f(m, \sigma_1) \quad \text{and} \quad \sigma_2 := \sigma_2(\sigma_1, m, h),$$

i.e. σ_2 depends only on the values σ_1, m, and h. Furthermore, a signature is independent of previous signatures.

[1] I.e., the number of issued coins ‘ $\leq (\log k)^{\alpha}$, where k is the security parameter and is some constant.

Notice that all signature schemes resulting from the transformation from [FS86] [2] applied to a (zero-knowledge) three-move identification protocol fall in this class.

6.1.2 Attacks

Two different types of attacks of an attacker $\mathcal{A}$ (who is a probabilistic polynomial-time Turing machine) were considered:

- the so-called *no-message attack*, in which $\mathcal{A}$ knows only the signer's public key and other public data; he may ask queries to the hash function f, but can't interact with the signer. The attacker's goal is to output a valid message-signature pair.
- the *adaptively-chosen-message attack*, which is maybe the strongest imaginable attack [3]; here $\mathcal{A}$ can ask the signer to sign arbitrary messages (chosen-message). He may use previous message-signature pairs to determine which message to let the signer sign next (adaptivity). The attacker's goal is to output a signature on a message the signer didn't sign.

Notice that these are the two extremes among the possible attacks; a no-message attacker is as restricted as can be, while an adaptively-chosen-message attacker is allowed to do almost anything. Security against one specific kind of attack only implies security against less powerful attacks. Thus, the term "security" should always be used in a precise manner, i.e. by speaking of "security against a ... attack".

6.1.3 The Forking Lemma

The forking lemma is the foundation for the proofs of security in [PS96a]. We'll first state it, then give an intuitive explanation of its meaning and finally present the proof.

Lemma 6.1 (The forking lemma). Consider a signature scheme as described above in the random oracle model. Let $\mathcal{A}$ be a probabilistic polynomial-time Turing machine, given only the public data as input. If $\mathcal{A}$ can find, with nonnegligible probability, a valid message-signature pair $(m, (\sigma_1, h, \sigma_2))$, then, with nonnegligible probability, a replay of this machine with the same random tape and a different oracle outputs another valid message-signature pair $(m, (\sigma_1, h', \sigma_2'))$ such that $h \neq h'$.

"Replay" means that the machine $\mathcal{A}$ is started again; as mentioned, this happens with the same inputs and the same random tape, but with a different oracle. If $\mathcal{A}$ has nonnegligible probability of success, then obviously with nonnegligible probability two runs of $\mathcal{A}$ will produce two signatures. The lemma states that with nonnegligible probability they will both be signatures on m of the form (σ_1, h, σ_2) and $(\sigma_1, h', \sigma_2')$ such that $h \neq h'$. This is due to the bifurcation of queries and responses sketched in Figure 6.1: During the first run, $\mathcal{A}$ asks Q queries $\mathcal{Q}_1, \ldots, \mathcal{Q}_Q$ and obtains responses $\rho_1, \ldots, \rho_Q$. During the replay, for some $\beta \in \{1, \ldots, Q\}$, the new oracle gives identical answers to the first $\beta - 1$ queries, but replies $\rho'_\beta \neq \rho_\beta$ to query $\mathcal{Q}_\beta$. From there on, $\mathcal{A}$ may ask new queries $\mathcal{Q}_{\beta+1}, \ldots, \mathcal{Q}_Q$, because these queries are chosen adaptively, i.e. depending on previous responses. This bifurcation leads to the name "forking lemma". With the two signatures, we'll be able to solve, in polynomial time, some hard problem.

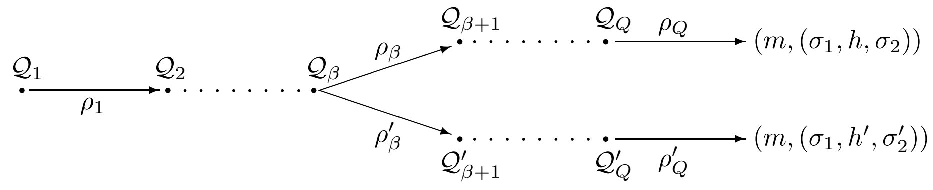


Figure 6.1: The forking lemma.

[2] As mentioned in Subsection 2.7.3, this was creating signature schemes from three-move identification protocols by replacing the random challenge c by $c := \mathcal{H}(m||a)$.

[3] If we exclude attacks where $\mathcal{A}$ forces the signer to reveal his secrets.

Before proceeding with the proof of the forking lemma, we state the "probabilistic lemma" from [PS96a] that will be useful in the proof. [PS97] contains an improved version of it called the "splitting lemma".

Lemma 6.2 (The probabilistic lemma). For sets X, Y, let $A \subseteq X \times Y$ be such that the probability $\mathrm{P}_{x,y}[(x, y) \in A] \geq \varepsilon$. Then there exists a subset $\Xi \subseteq X$ such that

1. $\mathrm{P}_x[x \in \Xi] \geq \varepsilon/2$,
2. for $a \in \Xi$: $\mathrm{P}_y[(a, y) \in A] \geq \varepsilon/2$.

This means that the set X can be split into two subsets Ξ, $X \backslash \Xi$ such that the nonnegligible [4] set Ξ consists of "good" x's which provide a nonnegligible probability over y that $(x, y) \in A$. We now move on to the proof of the forking lemma.

Proof of Lemma 6.1. $\mathcal{A}$ is a probabilistic polynomial-time Turing machine with a random tape ω. During the no-message attack, $\mathcal{A}$ asks Q distinct queries $\mathcal{Q}_1, \ldots, \mathcal{Q}_Q$ to the oracle f for some Q polynomial in k. Let $\rho_1, \ldots, \rho_Q$ be the corresponding answers of f. Randomly choosing f corresponds to a random choice of $\rho_1, \ldots, \rho_Q$. For a random choice of $\omega, \rho_1, \ldots, \rho_Q$, $\mathcal{A}$ outputs (with nonnegligible probability) a valid signature $(m, \sigma_1, h, \sigma_2)$. With overwhelming probability, the query (m, σ_1) is asked during a successful attack. Recall that $h := f(m, \sigma_1)$; since f is random, it is rather improbable that $\mathcal{A}$ can compute h correctly without ever asking $f(m, \sigma_1)$. To do so, $\mathcal{A}$ would have to guess h correctly among the 2^k possibilities. Hence, there exists a $\beta \in \{1, \ldots, Q\}$ such that the probability of success over $\omega, \rho_1, \ldots, \rho_Q$ with $\mathcal{Q}_\beta = (m, \sigma_1)$ is nonnegligible, say $1/P(n)$ for a polynomial P and n the length of the public key.

Using Lemma 6.2, we'll now fix the other variables. If we identify

- the set Ω of all possible random tapes ω with X,
- the set R of all possible response-tuples $(\rho_1, \ldots, \rho_Q)$ with Y,
- the set of successful tuples $(\omega, \rho_1, \ldots, \rho_Q)$ $\mathcal{S} \subseteq \Omega \times R$ with A,
- the probability $\mathrm{P}_{\omega,\rho_1,\ldots,\rho_Q}[(\omega, \rho_1, \ldots, \rho_Q) \in \mathcal{S}] = 1/P(n)$ with ε,

we know that for this β there is a nonnegligible subset $\Omega_\beta \subseteq \Omega$ of "good" random tapes ω. For $\mathcal{Q}_\beta = (m, \sigma_1)$ and $\omega \in \Omega_\beta$, the probability of success over the oracle responses $\rho_1, \ldots, \rho_Q$ is greater than $1/(2P(n))$:

$$\forall \omega \in \Omega_\beta : \mathrm{P}_{\rho_1,\ldots,\rho_Q}[(\omega, \rho_1, \ldots, \rho_Q) \in \mathcal{S}] \geq \frac{1}{2P(n)}.$$

Analogously, there exists a nonnegligible subset $R_{\beta,\omega}$ of "good" tuples $(\rho_1, \ldots, \rho_{\beta-1})$. Now, if we also choose a "good" tuple $(\rho_1, \ldots, \rho_{\beta-1})$, the probability of success of the attack over $\rho_\beta, \ldots, \rho_Q$ is greater than $1/(4P(n))$. Then, with such β, ω, and $(\rho_1, \ldots, \rho_{\beta-1})$, if we randomly choose $\rho_\beta, \ldots, \rho_Q$ and $\rho'_\beta, \ldots, \rho'_Q$, with a nonnegligible probability we obtain two valid signatures (σ_1, h, σ_2) and $(\sigma_1, h', \sigma'_2)$ on m. Since

- $h := f(m, \sigma_1)$ and $h' := f(m, \sigma'_1)$,
- f is random,
- each output value of f has probability $1/2^k$,

with nonnegligible probability (exactly $1 - 1/2^{k+1}$) we have $h \neq h'$.

Finally, if we randomly choose $\beta, \omega, \rho_1, \ldots, \rho_{\beta-1}, \rho_\beta, \ldots, \rho_Q$ and $\rho'_\beta, \ldots, \rho'_Q$ [5], then with nonnegligible probability we obtain two valid signatures (σ_1, h, σ_2) and $(\sigma_1, h', \sigma'_2)$ on m such that $h \neq h'$.

[4] With respect to its size, since $\mathrm{P}_x[\mathsf{x} \in \Xi] \geq$ "=2.

[5] $(\ '_1; : : : ; \ '_{\beta-1}) := (\ _1; : : : ; \ _{\beta-1})$.

6.1.4 Security Against No-Message Attacks

In [PS96a], the forking lemma was used to prove the security against no-message attacks of the signature scheme from [FS86] and of a modified ElGamal scheme. We'll do the same here for the Schnorr signature scheme as described in Subsection 2.5.4.

Theorem 6.3. Consider a no-message attack in the random oracle model. If an existential forgery (see Section 2.5) of a Schnorr signature is possible with nonnegligible probability, then the discrete logarithm problem can be solved in polynomial time.

Proof. Let $\mathcal{A}_1$ be a probabilistic polynomial-time attacker who can existentially forge Schnorr signatures with only the public data (group G of prime order q, generator g, public key $y := g^x$) as his input. According to the forking lemma, running $\mathcal{A}_1$ twice with different oracles $\mathcal{H}$, $\mathcal{H}'$ yields two valid Schnorr signatures (c, s), (c', s') on a message m such that the verification equations

$$c \overset{?}{=} \mathcal{H}(m||g^s y^c)$$

$$c' \overset{?}{=} \mathcal{H}'(m||g^{s'} y^{c'})$$

hold. One would like to have

$$g^s y^c = g^{s'} y^{c'}, \tag{6.1}$$

but this can't be seen as a direct consequence of the two verification equations, since $c \neq c'$ and $\mathcal{H} \neq \mathcal{H}'$. However, if we define the signatures as (a, c, s), (a, c', s') such that $c = \mathcal{H}(m||a)$ and $c' = \mathcal{H}'(m||a)$ [6], and the verification equations as

$$c \overset{?}{=} \mathcal{H}(m||a) \quad \text{and} \quad a \overset{?}{=} g^s y^c$$

$$c' \overset{?}{=} \mathcal{H}'(m||a) \quad \text{and} \quad a \overset{?}{=} g^{s'} y^{c'},$$

we directly get Equation 6.1 and can proceed to compute the discrete logarithm of y with respect to g:

$$\begin{array}{rrcl} & g^s y^c & = & g^{s'} y^{c'} \\ \Leftrightarrow & g^{s-s'} & = & y^{c'-c} \\ \Leftrightarrow & s - s' & = & \log_g y^{c'-c} \\ \Leftrightarrow & \dfrac{s-s'}{c'-c} & = & \log_g y. \end{array}$$

As one can see, (a, c, s) corresponds directly to (σ_1, h, σ_2) from Subsection 6.1.1. The "modification" of the Schnorr signature performed above actually isn't one, since all the equations already appear as such in the signature scheme. The signature in the form (c, s) is just a shorter variant of (a, c, s).

6.1.5 Security Against Adaptively-Chosen-Message Attacks

During an adaptively-chosen-message attack, the attacker $\mathcal{A}$ is allowed to ask the signer Σ to sign arbitrary messages. If $\mathcal{A}$ has nonnegligible probability of success, then if it is possible to simulate the signer by a simulator $\mathcal{S}$ who doesn't know the signer's secret key, the collusion of $\mathcal{S}$ and $\mathcal{A}$ will be able to forge a signature and, with the forking lemma, to solve the underlying hard problem.

Why would we want to simulate the signer and not just use the real one for our purposes? One reason is that it's rather unlikely that there's always a signer who will sign arbitrary messages for $\mathcal{A}$. Another reason is that as in Figure 6.2, $\mathcal{A}$ can be used as a "black box" [7] to create, together with $\mathcal{S}$, a new Turing machine $\mathcal{A}_+$ which, given only the public data as input, will be able to forge a signature and thus, with the forking lemma, to solve the hard problem. Hence, we obtain a machine whose correct function doesn't

[6] We can do this because according to the forking lemma, c and c' result from the same query to the two different oracles.

[7] This means that only $\mathcal{A}$'s input/output-behavior is known, i.e. we don't know anything about $\mathcal{A}$'s inner structure; the only constraint is that $\mathcal{A}$ is a probabilistic polynomial-time Turing machine.

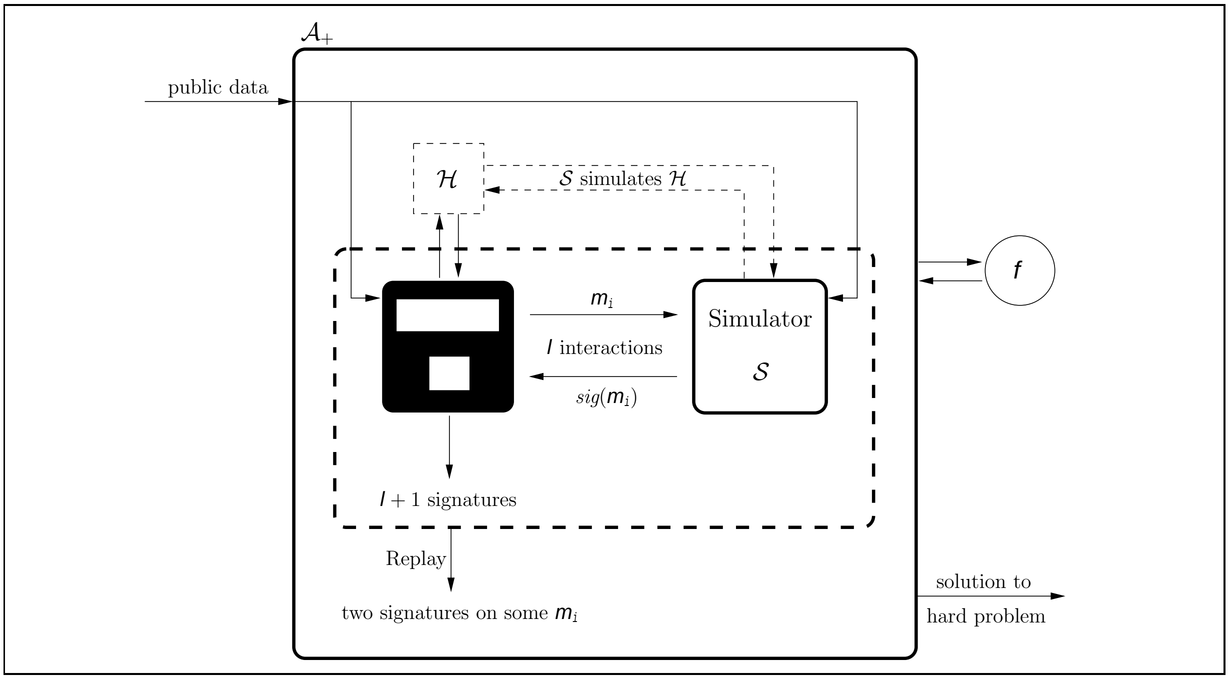


Figure 6.2: Usage of $\mathcal{A}$ as a black box.

depend on the cooperation of some other machine (like the signer). But the most important reason for using a simulator is that we can't apply the forking lemma with a real signer, as we can't "run him again with a different oracle". As $\mathcal{S}$ simulates the oracle $\mathcal{H}$ used by $\mathcal{A}$, this is no problem in the simulation.

One might also think of the possibility to let $\mathcal{S}$ choose his own secret key and to act as a "regular" signer; then, $\mathcal{A}_+$ would be able to solve the hard problem. This won't work because the instance of the hard problem depends on the secret key; e.g., the secret key being x with $y = g^x$ the corresponding public key, the hard problem would be the discrete logarithm problem, and the instance of it would be finding x, given only y. $\mathcal{A}_+$ would solve this instance, thus obtaining x. But since x was chosen by $\mathcal{S}$ (who is a part of $\mathcal{A}_+$), $\mathcal{A}_+$ wouldn't have gained any new knowledge. Hence, the public key must be chosen by some entity that is not part of $\mathcal{A}_+$.

$\mathcal{A}$ mustn't be able to distinguish $\mathcal{S}$ from the real signer, as $\mathcal{A}$'s behavior and thus the probability of success might change; this would forbid us to draw any useful conclusions. Formally, the views of $\mathcal{A}$ when interacting with the real signer and with $\mathcal{S}$ must be indistinguishable.

We'll now prove the security of the Schnorr signature scheme against adaptively-chosen-message attacks. The following lemma states that a Schnorr signer can be simulated.

Lemma 6.4. The signer in the Schnorr signature scheme can be simulated in an indistinguishable way by a probabilistic polynomial-time simulator $\mathcal{S}$.

Proof. Upon receiving a message m to sign, the simulator $\mathcal{S}$ proceeds as follows:

1. choose $c, s \in_R \mathbf{Z}_q$,
2. compute $a := g^s y^c$,
3. define [8] $\mathcal{H}(m||a) := c$.

Now, (a, c, s) is a valid signature on m. It is indistinguishable from the signature (a', c', s') an honest signer would have computed, since

[8] Recall that $\mathcal{S}$ simulates $\mathcal{H}$ and may therefore "define" its outputs.

- $a' := g^r$ for some random r, thus a' is random, just as a,
- $c' := \mathcal{H}(m||a')$ is random, just as c, because $\mathcal{H}$ is a random oracle and because a' is random,
- $s' := r - c'x$ is random due to the randomness [9] of r and c', just as s.

If $\mathcal{A}$ asks a query Q to $\mathcal{H}$, $\mathcal{S}$ lets $\mathcal{H}$ output $f(Q)$, where f is a random oracle $\mathcal{S}$ has access to. Like this, the simulated $\mathcal{H}$ is indistinguishable from a real random oracle, because its responses are either identical to those of the real random oracle f or defined during signature generation as $\mathcal{H}(m||a) := c$, where c is a random value. The only possibility for $\mathcal{A}$ to notice a difference is that there's a collision for $\mathcal{H}(Q)$, i.e. one of the following things happens:

- during signature generation, $\mathcal{S}$ defines $\mathcal{H}(m||a) := c$; for another signature, it happens that $g^{s'}h^{c'} = a$, and $\mathcal{H}(m||a)$ is redefined as $\mathcal{H}(m||a) := c' \neq c$. Then, one of the two signatures won't pass the verification and $\mathcal{A}$ notices that $\mathcal{H}$ has given two different responses to the same query,
- $\mathcal{S}$ defines $\mathcal{H}(m||a) := c$ during signature generation, but $\mathcal{A}$ has already asked $\mathcal{H}(Q) = \mathcal{H}(m||a)$ and obtained the response $f(Q) \neq c$ before. $\mathcal{A}$ then notices that $\mathcal{H}$ has given two different responses to the same query.

The first collision scenario occurs only with negligible probability $1/q$, as c, s are chosen randomly (by the way, an algorithm that efficiently found quadruples (c, s, c', s') such that $a = g^s h^c = g^{s'}h^{c'}$ and $(c, s) \neq (c', s')$ could compute $\log_g h$).

To state that also the second possibility for collisions has only negligible probability, we have to assume that before asking $\mathcal{S}$ to sign a message m, $\mathcal{A}$ doesn't perform a large number of queries $\mathcal{H}(m||\alpha)$ for different values $\alpha \in \mathbf{Z}_q$. Since the outputs of $\mathcal{H}$ are random, there is no point in asking such queries. If $\mathcal{A}$ nevertheless does so, there exists another attacker $\mathcal{A}'$ with the same probability of success who doesn't ask these queries. The probability doesn't change because there's no conceivable way how obtaining a large number of random values could help the attacker.

Theorem 6.5. Consider an adaptively-chosen-message attack in the random oracle model. If an existential forgery of a Schnorr signature is possible with nonnegligible probability, then the discrete logarithm problem can be solved in polynomial time.

Proof. Let $\mathcal{A}_2$ be an attacker who can existentially forge Schnorr signatures in an adaptively-chosen-message-attack against the signer. We can simulate the signer by a probabilistic polynomial-time simulator $\mathcal{S}$ in an indistinguishable way. According to the forking lemma, running $\mathcal{A}_2$ twice with different oracles $\mathcal{H}$, $\mathcal{H}'$ yields two valid Schnorr signatures (a, c, s), (a, c', s') on a message m such that the verification equations

$$c \stackrel{?}{=} \mathcal{H}(m||a) \quad \text{and} \quad a \stackrel{?}{=} g^s y^c$$
$$c' \stackrel{?}{=} \mathcal{H}'(m||a) \quad \text{and} \quad a \stackrel{?}{=} g^{s'} y^{c'},$$

hold. As in the no-message attack, we obtain

$$g^s y^c = g^{s'} y^{c'},$$

and can proceed to compute the discrete logarithm $\log_g y$.

6.2 Blind Signatures

As mentioned in Section 2.6, the definition of a successful forgery is different for blind signatures in comparison to normal (i.e. nonblind) signatures. Actually, a blind signature always constitutes an existential forgery in the classic sense, as it is a signature the signer didn't issue (in its unblind form).

For e-cash systems, the underlying blind signature scheme is required to prevent that anyone can obtain more valid signatures (i.e. coins) than were actually issued. This can be formalized as in the following definition.

[9] There is no correlation between r and c' although $c' := \mathcal{H}(m||g^r)$, because $\mathcal{H}$ is a random oracle.

Definition 6.1 **($(\ell, \ell+1)$-forgery).** Consider a blind signature scheme. For a positive integer ℓ polynomial in some security parameter of the scheme, an $(\ell, \ell+1)$-forgery occurs if a probabilistic polynomial-time Turing machine $\mathcal{A}$ outputs $\ell+1$ valid signatures after only ℓ interactions with the signer.

A blind signature scheme is as secure against no-message attacks as the underlying nonblind scheme. This can be seen by considering that in a no-message attack there is no interaction with the signer, and thus no need for blinding. Hence, blinding doesn't change the probability of success of a no-message attack.

The methods from the previous section to obtain proofs of security against adaptively-chosen-message attacks can't be directly used for blind signature schemes, because the message is hidden from the signer and the correctness of the resulting signature can't be enforced by simulating the oracle $\mathcal{H}$ appropriately [10]. Thus, the signer can't be simulated without the secret key. The solution is to use schemes that have certain properties; e.g., they allow more than one secret key for a given public key. We'll see how this works in the following.

6.2.1 Witness Indistinguishability

The notion of "witness indistinguishability" was introduced in [FS90] for identification protocols. The term "witness" comes from proof of knowledge protocols, where the prover wants to show that he knows a secret value x (without revealing x) such that some relation $R(x, y)$ holds for a value y known to both the prover and the verifier. The value x is called the witness. Identification protocols as the one of Schnorr described in Subsection 2.7.3 are proof of knowledge protocols with y the prover's public key, x the corresponding secret key, and $R(x, y) = \{(x, y) | y = g^x\}$ for a public generator g. By proving knowledge of x, the prover proves his identity [11].

Witness indistinguishability means that if during two protocol executions the prover uses two different witnesses x_1, x_2 such that $R(x_1, y)$ and $R(x_2, y)$, then the two different views of the verifier are indistinguishable. For Schnorr's identification protocol this is clearly impossible [12]. The protocol can, however, be easily modified [13] to provide witness indistinguishability.

The same modification can be applied to the Schnorr signature scheme resulting from the identification protocol. We could then speak of "secret key indistinguishability". Clearly, the blind variant of the Schnorr signature scheme can also be made "secret key indistinguishable". Such an adaptation was presented in [Oka93]; it is described in the following subsection.

6.2.2 Okamoto-Schnorr Blind Signatures

For a given security parameter k, primes p, q are chosen such that $q|(p-1)$ and $2^{k-1} < q \leq 2^k$, i.e. q divides $(p-1)$ and q is of "the right size" with respect to k. We know from Subsection 2.2.1 that the group $\mathbf{Z}_p^*$ has order $p-1$ and that there exists a unique subgroup $G \subseteq \mathbf{Z}_p^*$ of order q. Two elements $g, h \in G$ are chosen; they are both generators of G.

As in Schnorr's signature scheme, G is a finite cyclic group of order q with generators g, h such that computing discrete logarithms is infeasible. Let $(r, s) \in (\mathbf{Z}_q^* \times \mathbf{Z}_q^*)$ be the signer's secret key, and let $y := g^{-r}h^{-s}$ be the corresponding public key. Let $\mathcal{H}$ be a collision-resistant one-way hash function that maps $\{0,1\}^*$ to $\mathbf{Z}_q$.

If both participants follow the protocol in Figure 6.3, $(\alpha, \varepsilon, \rho, \sigma)$ is a valid Okamoto-Schnorr signature on m; the verification equations

$$\varepsilon \stackrel{?}{=} \mathcal{H}(m||\alpha)$$

[10] This can be easily seen by trying to prove the security of, e.g., the blind Schnorr scheme from Subsection 2.6.1. The problem is that when the verifier asks $\mathcal{H}(m||t)$, the simulator has to respond before getting to know in the next step (when he receives c', he can compute $= c - c'$). As he obtains too late and doesn't get at all, it's impossible for him to provide the verifier with a valid signature.

[11] He proves "I'm the one who knows the secret key x corresponding to public key y"; assuming that he's the only one who knows x, this proves his identity.

[12] In the protocol, $x_1 \neq x_2$ means $x_1 \neq x_2 \pmod{q}$ for $q = \mathrm{ord}(g)$, and thus $g^{x_1} \neq g^{x_2}$.

[13] The idea is to define the public key as $y := g^r h^s$ for two generators $g; h$ and a secret key consisting of the pair $(r; s)$.

$$\alpha \overset{?}{=} g^{\rho} h^{\sigma} y^{\varepsilon}$$

hold.

$$\begin{array}{ccc}
\mathcal{V} & & \Sigma \\
(y, m) & & (y, (r, s)) \\
\hline
 & & t, u \in_R \mathbf{Z}_q \\
 & & a := g^t h^u \pmod{p} \\
 & \xleftarrow{\qquad a \qquad} & \\
\beta, \gamma, \delta \in_R \mathbf{Z}_q & & \\
\alpha := a g^{\beta} h^{\gamma} y^{\delta} \pmod{p} & & \\
\varepsilon := \mathcal{H}(m||\alpha) & & \\
e := \varepsilon - \delta \pmod{q} & & \\
 & \xrightarrow{\qquad e \qquad} & \\
 & & R := t + er \pmod{q} \\
 & & S := u + es \pmod{q} \\
 & \xleftarrow{\qquad R, S \qquad} & \\
a \overset{?}{=} g^R h^S y^e \pmod{p} & & \\
\rho := R + \beta \pmod{q} & & \\
\sigma := S + \gamma \pmod{q} & & \\
\hline
(\alpha, \varepsilon, \rho, \sigma) & &
\end{array}$$

Figure 6.3: Blind Okamoto-Schnorr signature.

If the signer Σ knows another representation (r', s') of y with respect to (g, h) and executes the protocol once using (r, s), once using (r', s'), then the two views of the verifier $\mathcal{V}$ are indistinguishable.

In [PS96b], a proof of security against $(\ell, \ell + 1)$-forgery for this scheme is given; the authors state that it can be modified so as to cover other schemes that come from witness indistinguishable identification protocols. We'll take a look at this proof in the following subsection.

6.2.3 Security Against (‘; ‘ + 1)-Forgery

The main result of [PS96b] is the following theorem:

Theorem 6.6. Consider the Okamoto-Schnorr blind signature scheme in the random oracle model. If there is a probabilistic polynomial-time Turing machine $\mathcal{A}$ which can perform an $(\ell, \ell + 1)$-forgery with nonnegligible probability, then there is a probabilistic polynomial-time machine which can solve the discrete logarithm problem by doing a replay with $\mathcal{A}$.

This is true even if the ℓ interactions with the signer are executed in parallel. This means that $\mathcal{A}$ is allowed to initiate all interactions at arbitrary times and may suspend interactions and proceed with others arbitrarily. We suppose he may use information from one interaction in other interactions.

The proof of this theorem uses the probabilistic lemma (Lemma 6.2) and a modified version of the forking lemma (Lemma 6.1):

Lemma 6.7 (The forking lemma for blind signatures). Consider a blind signature scheme resulting from a witness indistinguishable identification protocol in the random oracle model. Let $\mathcal{A}$ be a probabilistic polynomial-time attacker who can perform an $(\ell, \ell + 1)$-forgery with nonnegligible probability. Randomly choose an index $j \in \{1, \ldots, Q\}$ with Q the number of queries $\mathcal{A}$ asks to the random oracle. Run $\mathcal{A}$ twice [14] with the same random tape but two different random oracles f and f' which provide

[14] Or a polynomial number of times; a precise bound on the necessary number of replays was investigated in [PS97].

identical answers for the first $j-1$ queries, but differ (at least) for the j-th query. With nonnegligible probability, the two different outputs of $\mathcal{A}$ reveal two different representations of some α_i with respect to g and h.

As in [PS96b], we'll first present a sketched proof of Theorem 6.6 which uses Lemma 6.7, and afterwards present the proof of Lemma 6.7.

Proof (sketch) of Theorem 6.6. With nonnegligible probability, after ℓ interactions with the authority and a polynomial number of queries to the random oracle, $\mathcal{A}$ outputs $\ell+1$ valid message-signature pairs. Then, according to Lemma 6.7, a replay of $\mathcal{A}$ with the same random tapes but oracles f, f' of the described type will yield two valid signatures $(\alpha_i, \rho_i, \sigma_i, \varepsilon_i)$, $(\alpha_i, \rho_i', \sigma_i', \varepsilon_i')$ on m_i and hence

$$\begin{aligned}\alpha_i &= g^{\rho_i} h^{\sigma_i} y^{\varepsilon_i} = g^{\rho_i - r\varepsilon_i} h^{\sigma_i - s\varepsilon_i}\\ &= g^{\rho_i'} h^{\sigma_i'} y^{\varepsilon_i'} = g^{\rho_i' - r\varepsilon_i'} h^{\sigma_i' - s\varepsilon_i'}.\end{aligned}$$

Now we have two representations of α_i with respect to (g, h) and can compute the discrete logarithm of h with respect to g as

$$\log_g h = \frac{\rho_i - r\varepsilon_i - (\rho_i' - r\varepsilon_i')}{\sigma_i' - s\varepsilon_i' - (\sigma_i - s\varepsilon_i)} = \frac{\rho_i - \rho_i' - r(\varepsilon_i + \varepsilon_i')}{\sigma_i' - \sigma_i - s(\varepsilon_i' + \varepsilon_i)}.$$

It is clear that we need the signer's secret key (r, s) for this; additionally, we haven't so far addressed the question of how to convince the signer to participate in the replay. The solution is that the previously used simulator now becomes a regular signer who cooperates with the attacker. Together, they can be used to construct a probabilistic polynomial-time Turing machine $\mathcal{A}_+$ that can compute discrete logarithms as in Figure 6.4. This is possible here (in contrast to the proof for nonblind signatures in the previous section) because the instance of the hard problem doesn't depend on the secret key, but only on the generators g, h which $\mathcal{A}_+$ doesn't choose. Thus, $\mathcal{A}_+$ really gains new knowledge ($\log_g h$).

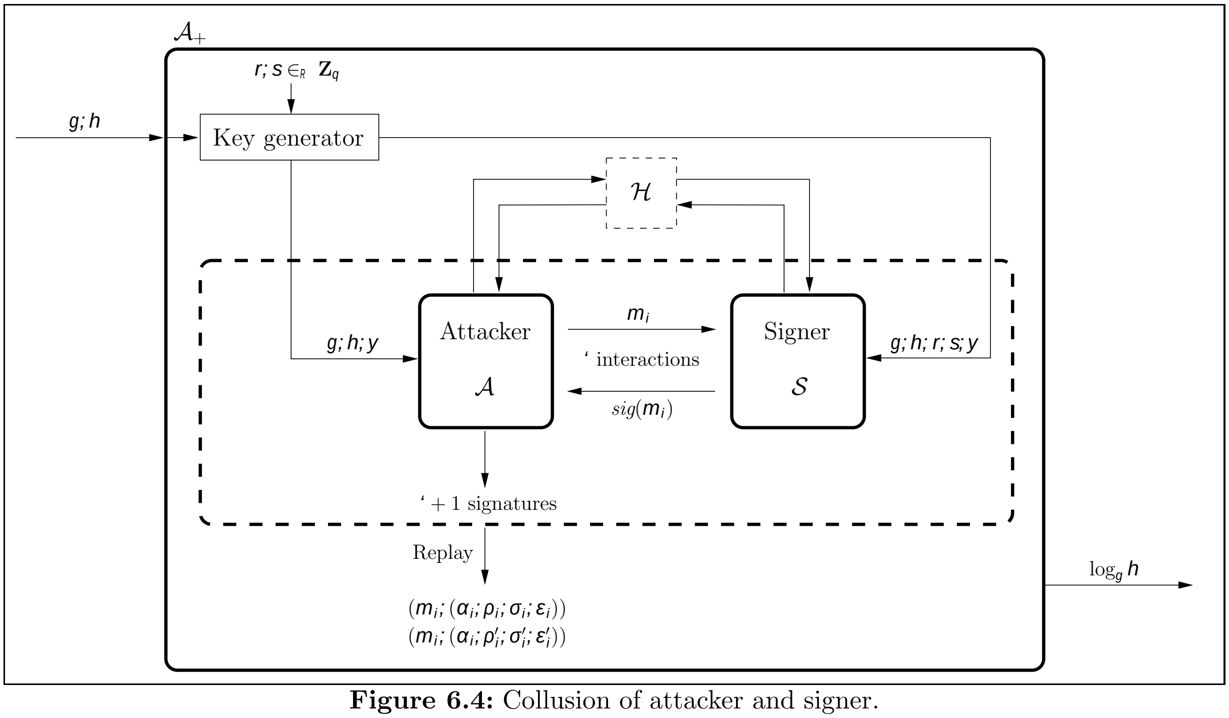


Figure 6.4: Collusion of attacker and signer.

Proof of Lemma 6.7. For the same reasons as in the proof of Lemma 6.1, we can assume that all the (m_i, α_i) have been queries to the oracle during the attack. Let the nonnegligible success probability of the attacker $\mathcal{A}$ be ε. Then, if we assume that among the total Q queries to the oracle, the indices

$Ind_1, \dots, Ind_{\ell+1} \in \{1, \dots, Q\}$ of the (m_i, α_i) are constant [15], the success probability might decrease down to $\rho := \varepsilon / Q^{\ell+1}$ (because of all the $Q^{\ell+1}$ different possible index tuples $(Ind_1, \dots, Ind_{\ell+1})$, we allow only one).

The signer's random tape is denoted by Ω; for $i \in \{1, \dots, \ell+1\}$, it determines the pairs (t_i, u_i) and these in turn the values $a_i := g^{t_i} h^{u_i}$. The distribution of (t_i, u_i, a_i) where t_i and u_i are random and $a_i = g^{t_i} h^{u_i}$ is the same as the distribution of (t_i, u_i, a_i) where t_i and a_i are random and u_i is the unique value such that $a_i = g^{t_i} h^{u_i}$. Thus, (t_i, u_i) will be replaced by (t_i, a_i) and, analogously, (r, s) will be replaced by (r, y) in the following.

To facilitate handling the various values involved, they are grouped under variables as follows:

- $\nu := (\omega, y, a_1, \dots, a_\ell)$, where ω denotes $\mathcal{A}$'s random tape,
- $\tau := (t_1, \dots, t_\ell)$.

We will denote by $\mathcal{S}$ the set of all successful quadruples (ν, r, τ, f), i.e. quadruples such that the $(\ell, \ell+1)$-forgery succeeds, where f is the random oracle. Then $\mathrm{P}_{\nu,r,\tau,f}[(\nu, r, \tau, f) \in \mathcal{S}] \geq \rho$.

We want to prove that the lemma holds, i.e. that after a replay [16], we obtain two valid signatures

$$(\alpha_i, \rho_i, \sigma_i, \varepsilon_i) \quad \text{and} \quad (\alpha_i, \rho_i', \sigma_i', \varepsilon_i')$$

on m_i. Their validity implies

$$\begin{aligned} \alpha_i &= g^{\rho_i} h^{\sigma_i} y^{\varepsilon_i} = g^{\rho_i - r\varepsilon_i} h^{\sigma_i - s\varepsilon_i} \\ &= g^{\rho_i'} h^{\sigma_i'} y^{\varepsilon_i'} = g^{\rho_i' - r\varepsilon_i'} h^{\sigma_i' - s\varepsilon_i'}. \end{aligned}$$

Moreover, we need $\rho_i - r\varepsilon_i \neq \rho_i' - r\varepsilon_i'$ in order to compute $\log_g h$.

For each $i \in \{1, \dots, \ell+1\}$, α_i can only depend on ν, r, τ and the first $Ind_i - 1$ answers of the oracle f. The question is whether the random variable $\chi_i = \rho_i - r\varepsilon_i$ depends on queries asked at steps Ind_i, $Ind_i + 1$ etc. Hopefully, this is the case and by using two oracles that give different responses from step Ind_i on, we'll obtain values χ_i, χ_i' such that $\chi_i \neq \chi_i'$.

We consider the most likely value taken by χ_i when (ν, r, τ) and the $Ind_i - 1$ first answers of f are fixed. We'll denote by f_i the restriction of f to the $Ind_i - 1$ first queries. Now, we define

$$\lambda_i(\nu, r, \tau, f_i, c) := \mathrm{P}_f\Big[\big(\chi_i(\nu, r, \tau, f) = c\big) \wedge \big((\nu, r, \tau, f) \in \mathcal{S}\big) \Big| f \text{ extends } f_i\Big].$$

Thus, $\lambda_i(\nu, r, \tau, f_i, c)$ indicates the probability, over all f that extend the fixed f_i, that the attack succeeds and that $\chi_i = c$. Next, we define

$$c_i(\nu, r, \tau, f_i) := c \text{ such that } \forall c' \neq c : \lambda_i(\nu, r, \tau, f_i, c) \geq \lambda_i(\nu, r, \tau, f_i, c').$$

Hence, c_i is defined as the value c such that the corresponding probability λ_i is maximal. Now, the "good" subset $\mathcal{G}$ of $\mathcal{S}$ is defined as

$$\mathcal{G} := \big\{(\nu, r, \tau, f) \in \mathcal{S} \mid \forall i \in \{1, \dots, \ell+1\} : \chi_i(\nu, r, \tau, f) = c_i(\nu, r, \tau, f_i)\big\},$$

and the "bad" subset $\mathcal{B} := \mathcal{S} \backslash \mathcal{G}$ as its complement in $\mathcal{S}$.

It's important to understand what these subsets are: $\mathcal{G}$ contains the tuples (ν, r, τ, f) such that for all $i \in \{1, \dots, \ell+1\}$ the variable $\chi_i = \rho_i - r\varepsilon_i$ takes the value with the maximal probability, i.e. c_i. For each tuple $(\nu, r, \tau, f) \in \mathcal{B}$, there is at least one $i \in \{1, \dots, \ell+1\}$ such that $\chi_i \neq c_i$.

If we would succeed in finding $(\nu, r, \tau, f) \in \mathcal{G}$ and $(\nu, r, \tau, f') \in \mathcal{B}$ such that f and f' output the same responses to the first $Ind_i - 1$ queries and that

$$\chi_i(\nu, r, \tau, f') \neq c_i(\nu, r, \tau, f_i') = c_i(\nu, r, \tau, f_i) = \chi_i(\nu, r, \tau, f) \text{ }^{17},$$

[15] I.e., $(m_j;\ \ _j)$ is always the Ind_j-th query for all $j \in \{1; : : : ; ` + 1\}$.

[16] In [PS97]: after a polynomial number of replays.

[17] The last two equalities hold since f and f' have identical outputs on the first $Ind_i - 1$ queries and because $(\ \ ; r;\ \ ; f) \in \mathcal{G}$.

then we would have a bifurcation at position i such that $\chi_i(\nu, r, \tau, f) \neq \chi_i(\nu, r, \tau, f')$ and thus, by definition, $\rho_i - r\varepsilon_i \neq \rho'_i - r\varepsilon'_i$.

The rest of the proof shows that if we randomly choose (ν, r, τ, f) and a forking index i, then a replay with an oracle $f' \neq f$ such that $f'_i = f_i$ (and everything else unchanged) gives us, with nonnegligible probability, the desired representations of α_i.

First, a one-to-one mapping, i.e. a bijective function, from $\mathcal{S}$ onto $\mathcal{S}$ is presented; the special properties of this mapping allow us to estimate the size of $\mathcal{B}$, which will then be used to show the success of forking.

Definition 6.2. We denote by Φ the transformation which maps any tuple (ν, r, τ, f) to $(\nu, r+1, \tau-e, f)$, where $\tau - e := (t_1 - e_1, \ldots, t_\ell - e_\ell)$; the values e_i are the challenges sent to the signer.

The transformation Φ has some useful properties:

Lemma 6.8. The views of the attacker in the two protocol executions corresponding to (ν, r, τ, f) and $\Phi(\nu, r, \tau, f)$ are identical, i.e. outputs are the same.

Proof. Let $r' := r + 1$ and $\tau' := \tau - e$. The answers of the oracle f are unchanged, and the responses of the signer become

$$R'_i(r', t'_i, e_i) = t'_i + r'e_i = (t_i - e_i) + (r+1)e_i = t_i + re_i = R_i(r, t_i, e_i).$$

This gives us the following corollary:

Corollary 6.9. The transformation Φ is a one-to-one mapping from $\mathcal{S}$ onto $\mathcal{S}$.

The following lemma shows that if we choose $(\nu, r, \tau, f) \in \mathcal{G}$, then $\Phi(\nu, r, \tau, f) \in \mathcal{B}$, except for a negligible part.

Lemma 6.10. For fixed (ν, r, τ), the probability

$$\mathrm{P}_f\big[(\nu, r, \tau, f) \in \mathcal{G} \wedge \Phi(\nu, r, \tau, f) \in \mathcal{G}\big]$$

over f is bounded by $1/q$.

Proof. Assume the opposite, i.e. that (for fixed (ν, r, τ))

$$\mathrm{P}_f\big[(\nu, r, \tau, f) \in \cup_{e_1,\ldots,e_\ell} Y(e_1, \ldots, e_\ell)\big] > \frac{1}{q}, \tag{6.2}$$

where the set $Y(e_1, \ldots, e_\ell)$ is defined as

$$Y(e_1, \ldots, e_\ell) := \big\{(\nu, r, \tau, f) \big| (\nu, r, \tau, f) \in \mathcal{G} \wedge \Phi(\nu, r, \tau, f) \in \mathcal{G} \wedge E = (e_1, \ldots, e_\ell)\big\},$$

with E denoting the ℓ-tuple of questions asked to the signer. Each e_i can have q different values, hence there are q^ℓ different possible tuples E. Therefore, with Equation 6.2, at least one ℓ-tuple $(e_1, \ldots, e_\ell)$ exists such that

$$\mathrm{P}_f\big[(\nu, r, \tau, f) \in Y(e_1, \ldots, e_\ell)\big] > \frac{1}{q^{\ell+1}}. \tag{6.3}$$

As there are at least $\ell + 1$ different queries to f (the (m_i, α_i) for $i \in \{1, \ldots, \ell+1\}$), we can distinguish at least $q^{\ell+1}$ different oracles f. Furthermore, we can treat oracles that have identical responses to the queries (m_i, α_i) but differ for some other queries as equal. This is because responses to queries not of the form (m_i, α_i) are random values that probably are of no relevance to the attack. Hence, Equation 6.3 implies that there are at least two different oracles f and f' in $Y(e_1, \ldots, e_\ell)$, in the sense that for some $j \in \{1, \ldots, \ell+1\}$, $f(m_j, \alpha_j) \neq f'(m_j, \alpha_j)$. To see this, assume that there's at most one such oracle. Then, the probability on the left side of Equation 6.3 is

$$\frac{\text{number of oracles } f \in Y(e_1, \ldots, e_\ell) = 1}{\text{total number of oracles} \geq q^{\ell+1}} \leq \frac{1}{q^{l+1}},$$

which violates the equation.

Let i be the smallest index j such that $f(m_j, \alpha_j) \neq f'(m_j, \alpha_j)$. Then $f_i = f'_i$ and $\varepsilon_i \neq \varepsilon'_i$. Furthermore, by definition of $Y(e_1, \ldots, e_\ell)$ we have $(\nu, r, \tau, f) \in \mathcal{G}$, $\Phi(\nu, r, \tau, f) \in \mathcal{G}$, $(\nu, r, \tau, f') \in \mathcal{G}$, and $\Phi(\nu, r, \tau, f') \in \mathcal{G}$.

According to Lemma 6.8 and by definition of $\mathcal{G}$,

$$\begin{aligned}
c_i(\nu, r, \tau, f_i) &= \rho_i(\nu, r, \tau, f) - r\varepsilon_i \\
&= \rho_i\big(\Phi(\nu, r, \tau, f)\big) - r\varepsilon_i = c_i(\nu, r+1, \tau - e, f_i) + \big((r+1) - r\big)\varepsilon_i \\
c_i(\nu, r, \tau, f'_i) &= \rho_i(\nu, r, \tau, f') - r\varepsilon'_i \\
&= \rho_i\big(\Phi(\nu, r, \tau, f')\big) - r\varepsilon'_i = c_i(\nu, r+1, \tau - e, f'_i) + \big((r+1) - r\big)\varepsilon'_i.
\end{aligned}$$

Now, $f_i = f'_i$ implies $c_i(\nu, r, \tau, f_i) = c_i(\nu, r, \tau, f'_i)$; by the equations above then $\varepsilon_i = \varepsilon'_i$, which is a contradiction. Hence, our initial assumption was wrong and the lemma holds.

By making the sum over all triples (ν, r, τ) and by using Corollary 6.9 and Lemma 6.10, we obtain

$$\begin{aligned}
\mathrm{P}[\mathcal{G}] &= \mathrm{P}_{\nu,r,\tau,f}\big[(\nu, r, \tau, f) \in \mathcal{G} \wedge \Phi(\nu, r, \tau, f) \in \mathcal{G}\big] \\
&\quad + \mathrm{P}_{\nu,r,\tau,f}\big[(\nu, r, \tau, f) \in \mathcal{G} \wedge \Phi(\nu, r, \tau, f) \in \mathcal{B}\big] \\
&\leq \frac{1}{q} + \mathrm{P}_{\nu,r,\tau,f}\big[(\nu, r, \tau, f) \in \mathcal{G} \wedge \Phi(\nu, r, \tau, f) \in \mathcal{B}\big] \\
&\leq \frac{1}{q} + \mathrm{P}_{\nu,r,\tau,f}\big[\Phi(\nu, r, \tau, f) \in \mathcal{B}\big] \\
&\leq \frac{1}{q} + \mathrm{P}[\mathcal{B}].
\end{aligned}$$

With $\mathrm{P}[\mathcal{G}] = \mathrm{P}[\mathcal{S}] - \mathrm{P}[\mathcal{B}]$, we obtain $\mathrm{P}[\mathcal{B}] \geq (\mathrm{P}[\mathcal{S}] - 1/q)/2$. Since $1/q$ is negligible with respect to $\mathrm{P}[\mathcal{S}]$ for sufficiently large q, we have $\mathrm{P}[\mathcal{B}] \geq \mathrm{P}[\mathcal{S}]/3 \geq \rho/3$. We will use this probability to show the success of forking.

$$\begin{aligned}
\frac{\rho}{3} \leq \mathrm{P}[\mathcal{B}] &= \mathrm{P}_{\nu,r,\tau,f}\big[(\nu, r, \tau, f) \in \mathcal{S} \wedge \exists i : \chi_i(\nu, r, \tau, f) \neq c_i(\nu, r, \tau, f_i)\big] \\
&\leq \sum_{i=1}^{\ell+1} \mathrm{P}_{\nu,r,\tau,f}\big[(\nu, r, \tau, f) \in \mathcal{S} \wedge \chi_i(\nu, r, \tau, f) \neq c_i(\nu, r, \tau, f_i)\big].
\end{aligned}$$

Then, a value k exists such that

$$\mathrm{P}_{\nu,r,\tau,f}\big[(\nu, r, \tau, f) \in \mathcal{S} \wedge \chi_k(\nu, r, \tau, f) \neq c_k(\nu, r, \tau, f_k)\big] \geq \frac{\rho}{3(\ell+1)}.$$

Let us randomly choose the forking index $i \in \{1, \ldots, \ell+1\}$. With probability of at least $1/(\ell+1)$, we have guessed $i = k$. Given this, Lemma 6.2 ensures that there is a set X of quadruples (ν, r, τ, f_i) such that

1. $\mathrm{P}_{\nu,r,\tau,f}\big[(\nu, r, \tau, f_i) \in X\big] \geq \frac{\rho}{6(\ell+1)}$,
2. for all $(\nu, r, \tau, f_i) \in X$, $\mathrm{P}_f\big[(\nu, r, \tau, f) \in \mathcal{S} \wedge \chi_i \neq c_i \big| f \text{ extends } f_i\big] \geq \frac{\rho}{6(\ell+1)}$.

Let us choose a random quadruple (ν, r, τ, f). With probability of at least $\rho/6(\ell+1)$, we have $(\nu, r, \tau, f_i) \in X$. Given this, again with probability of at least $\rho/6(\ell+1)$, we have $(\nu, r, \tau, f) \in \mathcal{S}$ and $\chi_i(\nu, r, \tau, f) \neq c_i(\nu, r, \tau, f_i)$. Thus, the probability that all three conditions

- $(\nu, r, \tau, f) \in \mathcal{S}$,
- $(\nu, r, \tau, f_i) \in X$,
- $\chi_i(\nu, r, \tau, f) \neq c_i(\nu, r, \tau, f_i)$

hold is at least $\big(\rho/6(\ell+1)\big)^2$. Let's assume the conditions hold, and let's denote by d the value $\chi_i(\nu, r, \tau, f)$ and by c the value $c_i(\nu, r, \tau, f_i)$.

Now, we'll show that if we replay with another random oracle f' such that $f'_i = f_i$ and everything else unchanged, we have

$$\mathrm{P}_{f'}\big[(\nu, r, \tau, f') \in \mathcal{S} \wedge \chi_i(\nu, r, \tau, f') \neq d \big| f' \text{ extends } f_i\big] \geq \rho/12(\ell+1).$$

This is the probability that the replay will succeed and that $\chi_i(\nu, r, \tau, f') \neq \chi_i(\nu, r, \tau, f)$; then, we have two representations of α_i with respect to g and h.

There are two cases to be considered, relatively to the value $\lambda_i(\nu, r, \tau, f_i, d)$:

- $\lambda_i(\nu, r, \tau, f_i, d) \geq \rho/12(\ell+1)$: by definition of c_i, we know that $\lambda_i(\nu, r, \tau, f_i, c) \geq \rho/12(\ell+1)$ (recall that c_i is the value c such that the corresponding λ_i is maximal). Since $c \neq d$, we have the claimed probability.

- $\lambda_i(\nu, r, \tau, f_i, d) < \rho/12(\ell+1)$:

$$\lambda_i(\nu, r, \tau, f_i, d) + \mathrm{P}_{f'}\big[(\nu, r, \tau, f') \in \mathcal{S} \wedge \chi_i(\nu, r, \tau, f') \neq d \big| f' \text{ extends } f_i\big]$$
$$= \mathrm{P}_{f'}\big[(\nu, r, \tau, f') \in \mathcal{S} \big| f' \text{ extends } f_i\big]$$
$$\geq \mathrm{P}_{f'}\big[(\nu, r, \tau, f') \in \mathcal{S} \wedge \chi_i(\nu, r, \tau, f') \neq c \big| f' \text{ extends } f_i\big] \geq \frac{\rho}{6(\ell+1)}$$
$$\Rightarrow \mathrm{P}_{f'}\big[(\nu, r, \tau, f') \in \mathcal{S} \wedge \chi_i(\nu, r, \tau, f') \neq d \big| f' \text{ extends } f_i\big] \geq \frac{\rho}{6(\ell+1)} - \lambda_i(\nu, r, \tau, f_i, d) > \frac{\rho}{12(\ell+1)}.$$

Global Complexity of the Reduction

The overall probability of success is at least

$$\frac{1}{\ell+1} \times \left(\frac{\rho}{6(\ell+1)}\right)^2 \times \frac{\rho}{12(\ell+1)} = \frac{1}{2(\ell+1)} \times \left(\frac{1}{6(\ell+1)} \times \frac{\varepsilon}{Q^{\ell+1}}\right)^3,$$

where $\rho = \varepsilon/Q^{\ell+1}$ with ε the probability of success of an $(\ell, \ell+1)$-forgery and Q the number of queries to the random oracle.

6.3 Polynomially Many Blind Signatures

In [Poi98], a generic transformation applicable to some blind signature schemes was presented; it basically implements a cut-and-choose-like approach by inserting a "checker" $\mathcal{C}$ between the signer and the verifier whose task it is to control the verifier's honest behavior. As will be seen, in the final scheme this task is performed by the signer himself, i.e. there is no need for a real third party. Regarding $\mathcal{C}$ as an autonomous participant who might collude with the signer or the verifier (and potential attacker) serves in proving the claimed security.

The transformation was applied to the Okamoto-Schnorr blind signature scheme; the security of the resulting scheme was presented step by step: if a probabilistic polynomial-time attacker can forge a signature after polynomially many interactions with the signer, then a probabilistic polynomial-time attacker against the basic Okamoto-Schnorr scheme can be constructed which succeeds after logarithmically many interactions. This latter attack is the one that was proven in [PS96b] to be as hard as the discrete logarithm problem.

Recall that the Okamoto-Schnorr signature scheme is secure in the random oracle model only if the number ℓ of signatures is poly-logarithmically bounded.

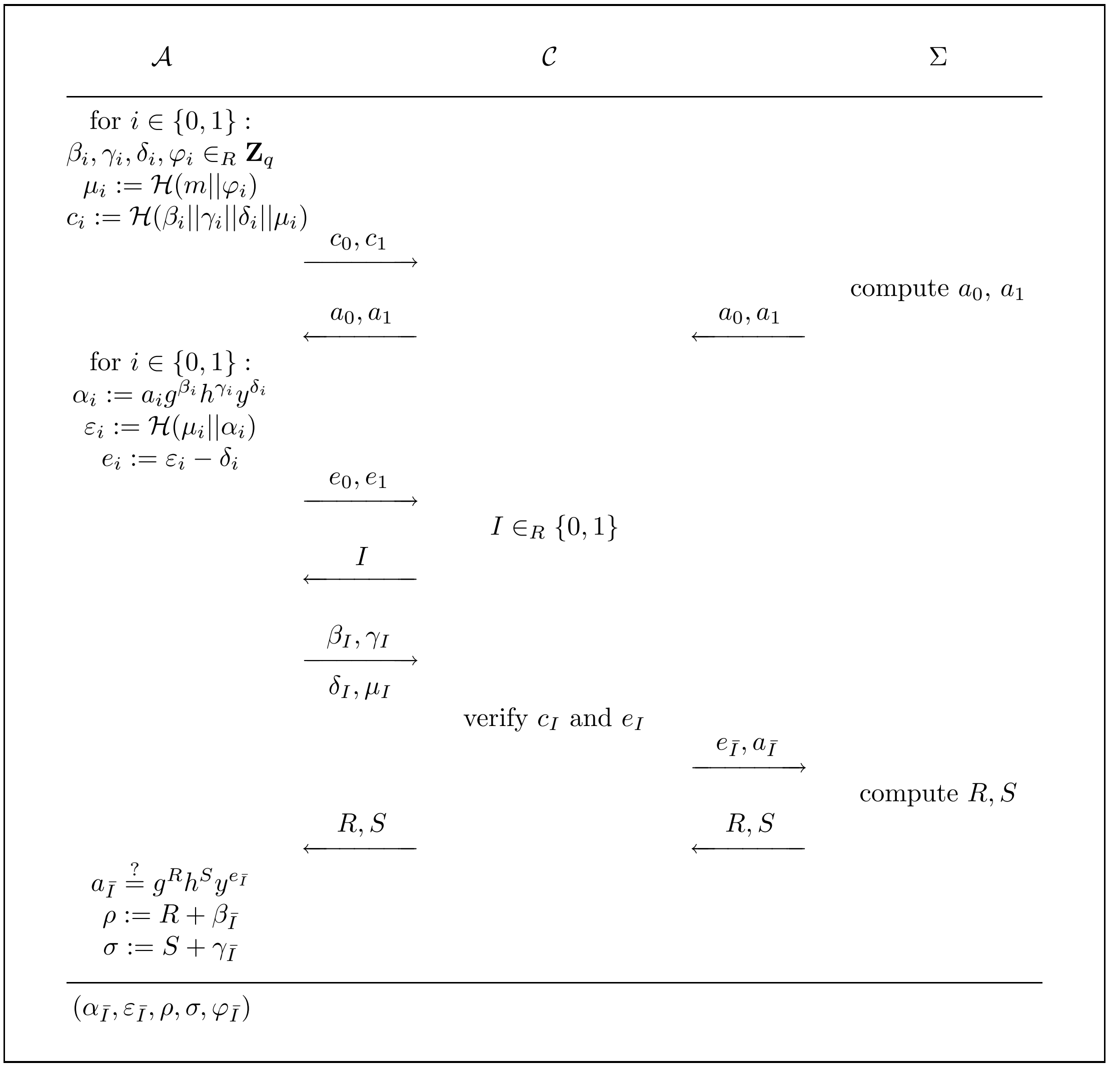

$\mathcal{A}$
$\mathcal{C}$
Σ
for $i \in \{0,1\}$:
$\beta_i, \gamma_i, \delta_i, \varphi_i \in_R \mathbf{Z}_q$
$\mu_i := \mathcal{H}(m||\varphi_i)$
$c_i := \mathcal{H}(\beta_i||\gamma_i||\delta_i||\mu_i)$
c_0, c_1
compute a_0, a_1
a_0, a_1
a_0, a_1
for $i \in \{0,1\}$:
$\alpha_i := a_i g^{\beta_i} h^{\gamma_i} y^{\delta_i}$
$\varepsilon_i := \mathcal{H}(\mu_i||\alpha_i)$
$e_i := \varepsilon_i - \delta_i$
e_0, e_1
$I \in_R \{0,1\}$
I
β_I, γ_I
δ_I, μ_I
verify c_I and e_I
$e_{\bar{I}}, a_{\bar{I}}$
compute R, S
R, S
R, S
$a_{\bar{I}} \overset{?}{=} g^R h^S y^{e_{\bar{I}}}$
$\rho := R + \beta_{\bar{I}}$
$\sigma := S + \gamma_{\bar{I}}$
$(\alpha_{\bar{I}}, \varepsilon_{\bar{I}}, \rho, \sigma, \varphi_{\bar{I}})$

Figure 6.5: The transformed Okamoto-Schnorr scheme.

6.3.1 Adding the Checker

To circumvent the restriction on the number ℓ of signatures, a checker $\mathcal{C}$ is inserted between the signer Σ and the verifier (and potential attacker) $\mathcal{A}$. The application of this transformation to the Okamoto-Schnorr scheme is presented in Figure 6.5: $\mathcal{A}$ chooses two tuples of blinding factors $(\beta_i, \gamma_i, \delta_i, \varphi_i)$ for $i \in \{0,1\}$ and sends the commitments c_0, c_1 to $\mathcal{C}$. Note that the random value φ_i is used to commit to m (otherwise $\mathcal{C}$ would see m and the signature would no longer be blind); $\mathcal{A}$ will end up with a signature on $\mu_{\bar{I}} := \mathcal{H}(m||\varphi_{\bar{I}})$ for a value $\bar{I}$ [18].

$\mathcal{C}$ initiates two parallel executions of the Okamoto-Schnorr blind signature protocol with Σ and receives the commitments a_0, a_1 which are forwarded to $\mathcal{A}$. $\mathcal{A}$ blinds them to α_0, α_1, computes the values ε_0, ε_1, and the challenges e_0, e_1 which are sent to $\mathcal{C}$. Now $\mathcal{C}$ randomly chooses $I \in \{0,1\}$, asks $\mathcal{A}$ for the values $\beta_I, \gamma_I, \delta_I, \mu_I$, and upon $\mathcal{A}$'s response checks the construction of c_I and e_I by verifying

$$c_I \overset{?}{=} \mathcal{H}(\beta_I||\gamma_I||\delta_I||\mu_I)$$

$$e_I \overset{?}{=} \mathcal{H}(\mu_I||a_I g^{\beta_I} h^{\gamma_I} y^{\delta_I}) - \delta_I .$$

$\mathcal{C}$ then asks $e_{\bar{I}}, a_{\bar{I}}$ to Σ. Observe that in [Poi98] $a_{\bar{I}}$ was omitted and Σ couldn't tell which of the two

[18] For $I \in \{0; 1\}$, $\bar{I}$ denotes $1 - I$.

protocol execution instances is the one to proceed with; however, later on, when Σ and $\mathcal{C}$ become one party, this is irrelevant. $\mathcal{C}$ obtains the response (R, S) which is forwarded to $\mathcal{A}$; $\mathcal{A}$ verifies its correctness and computes ρ, σ, thus obtaining the signature $(\alpha_{\bar{I}}, \varepsilon_{\bar{I}}, \rho, \sigma)$ on $\mu_{\bar{I}} := \mathcal{H}(m||\varphi_{\bar{I}})$. It is valid since

$$\alpha_{\bar{I}} = g^{\rho} h^{\sigma} y^{\varepsilon_{\bar{I}}}$$

$$\varepsilon_{\bar{I}} = \mathcal{H}(\mu_{\bar{I}}||\alpha_{\bar{I}}).$$

6.3.2 Security of the Scheme with Checker

In the following, we'll sketch the proof of security against coin forgery from [Poi98]. Afterwards, we'll take a look at the blindness of the scheme; a slight modification is needed to ensure it.

Security Against $(\ell, \ell+1)$-Forgery

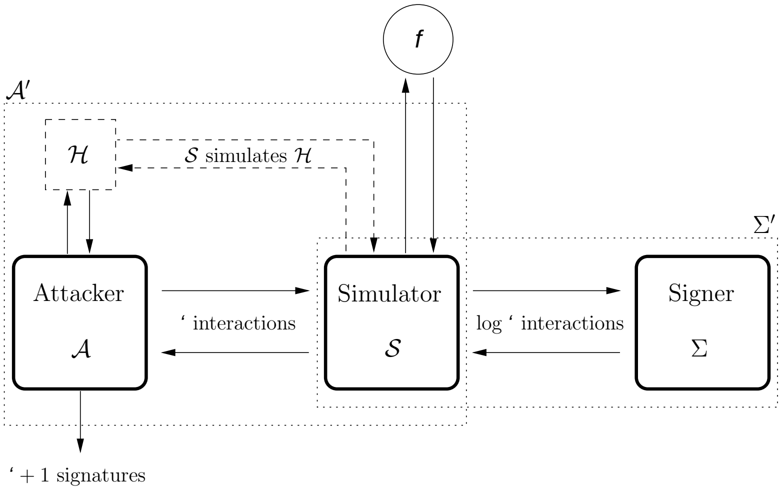


Figure 6.6: Simulation of the checker $\mathcal{C}$.

For $\mathcal{A}$, the signing party is constituted by the collusion of Σ and $\mathcal{C}$, denoted by Σ'. To prove the claimed security of the transformed scheme, Pointcheval first assumed that $\mathcal{A}$ is capable of obtaining $\ell+1$ signatures after ℓ interactions with Σ' with probability ε, where ℓ is polynomial in some security parameter. Then it was proven that if ε is nonnegligible, then a simulator $\mathcal{S}$ for $\mathcal{C}$ [19] as in Figure 6.6 exists such that

- $\mathcal{A}$ can't distinguish $\mathcal{S}$ from $\mathcal{C}$,
- during ℓ executions of the transformed scheme between $\mathcal{A}$ and $\mathcal{S}$, $\mathcal{S}$ initiates 2ℓ interactions with Σ, but completes only λ of them, where λ is logarithmically bounded; $\mathcal{S}$ computes the responses for the remaining $\ell - \lambda$ interactions by himself,
- among the $\ell+1$ apparently valid signatures obtained by $\mathcal{A}$, at least $\lambda+1$ are really valid, while the $\ell - \lambda$ signatures resulting from responses simulated by $\mathcal{S}$ are valid with respect to $\mathcal{H}$, and hence seem valid to $\mathcal{A}$; however, they are not valid with respect to f.

Then, the collusion of $\mathcal{S}$ and $\mathcal{A}$ can forge a signature in the basic Okamoto-Schnorr scheme after only λ interactions with Σ. But this was previously proven to be impossible unless solving the discrete logarithm problem is feasible.

[19] $\mathcal{S}$ controls the hash function $\mathcal{H}$ used by $\mathcal{A}$ and has access to the random oracle f.

Blindness

The scheme as proposed in [Poi98] actually isn't blind, but this can be achieved with a slight modification. We'll first see how signatures can be linked to corresponding protocol views:

Let the signer Σ' have a signature $(\alpha_{\bar{I}}, \varepsilon_{\bar{I}}, \rho, \sigma, \varphi_{\bar{I}})$ on a message m; he wants to find out during which execution of the signature protocol this signature was created. We suppose that Σ' has stored the views of all protocol executions. Now he can efficiently determine whether a view

$$\mathit{view} := (c_0, c_1, a_0, a_1, e_0, e_1, I, \beta_I, \gamma_I, \delta_I, \mu_I, R, S)$$

corresponds to the given signature as follows:

1. set
   $\delta_{\bar{I}} := \varepsilon_{\bar{I}} - e_{\bar{I}}$
   $\beta_{\bar{I}} := \rho - R$
   $\gamma_{\bar{I}} := \sigma - S,$
2. compute
   $\mu_{\bar{I}} := \mathcal{H}(m||\varphi_{\bar{I}})$
   $c := \mathcal{H}(\beta_{\bar{I}}||\gamma_{\bar{I}}||\delta_{\bar{I}}||\mu_{\bar{I}}),$
3. check $c \overset{?}{=} c_{\bar{I}}$.

This problem can be solved by "hiding" the values c_i by use of additional blinding factors ν_i, i.e. by letting $\mathcal{A}$ compute $c_i := \mathcal{H}(\beta_i||\gamma_i||\delta_i||\mu_i||\nu_i)$. In addition, after receiving I from Σ', $\mathcal{A}$ sends ν_I together with the other blinding factors $(\beta_{\bar{I}}, \gamma_{\bar{I}}, \delta_{\bar{I}}, \mu_{\bar{I}})$. The verification of c_I is then performed by checking

$$c_I \overset{?}{=} \mathcal{H}(\beta_{\bar{I}}||\gamma_{\bar{I}}||\delta_{\bar{I}}||\mu_{\bar{I}}||\nu_I).$$

We don't show the modified protocol here, as the changes will be reflected when we use it as a building block in the following section. We believe that our modification doesn't change the security of the scheme against $(\ell, \ell+1)$-forgery.

6.4 An On-line Scheme

Building an on-line payment system based on the results from [Poi98] is straightforward: a coin is represented by a random number together with a signature on it obtained by the transformed Okamoto-Schnorr blind signature scheme. Double-spending is no issue in such an on-line system, but the anonymity of it makes user blackmailing remain a problem. Therefore, implementing escrowing is investigated in the following: first, using a trustee as in Chapter 4; afterwards, in the form of self-escrowing as in Chapter 5.

The first solution that comes to mind would be to add anonymity revocation in the modular manner from [FTY98], especially as self-escrowing based on it has already been dealt with. However, this requires the underlying signature scheme to be a restrictive one, which isn't the case with the scheme presented above.

6.4.1 Implementing Escrowing with Trustee

The approach chosen here is to merge the scheme from [CMS96] presented in Subsection 4.3.1 with the transformed Okamoto-Schnorr signature scheme, thus obtaining a system in which, hopefully, coins are as hard to forge as signatures from [Poi98] and that is as safe as the one from [CMS96] under all other aspects. Additionally, self-escrowing and off-line payments might be implementable in the same way as for the scheme from [CMS96].

First, we modify the protocol of Figure 6.3; the result resembles the protocol “P” from [CMS96]. That protocol is a modification of the Schnorr blind signature protocol from figure 2.3, which in turn is similar to the Okamoto-Schnorr scheme. Therefore, the modifications to the Okamoto-Schnorr scheme were chosen in perfect analogy to the modifications that transform the Schnorr scheme into protocol “P”. Hopefully, they don’t affect the security of the signature scheme.

$\mathcal{V}$		Σ
$(y, m, h_{w_1}, h_{w_2}, \phi)$		$(y, (r, s), h_{w_1}, h_{w_2})$
		$z_w := h_{w_1}^{-r} h_{w_2}^{-s}$ $t, u \in_R \mathbf{Z}_q$ $a_g := g^t h^u$ $a_h := h_{w_1}^t h_{w_2}^u$
	$\xleftarrow{z_w, a_g, a_h}$	
$\beta, \gamma, \delta \in_R \mathbf{Z}_q$ $h_{p_1} := h_{w_1}^{\phi}$ $h_{p_2} := h_{w_2}^{\phi}$ $z_p := z_w^{\phi}$ $\alpha_g := a_g g^{\beta} h^{\gamma} y^{\delta}$ $\alpha_h := a_h^{\phi} h_{p_1}^{\beta} h_{p_2}^{\gamma} z_p^{\delta}$ $\varepsilon := \mathcal{H}(m \vert\vert \alpha_g \vert\vert \alpha_h)$ $e := \varepsilon - \delta$		
	$\xrightarrow{e}$	
		$R := t + er$ $S := u + es$
	$\xleftarrow{R, S}$	
$a_g \stackrel{?}{=} g^R h^S y^e$ $a_h \stackrel{?}{=} h_{w_1}^R h_{w_2}^S z_w^e$ $\rho := R + \beta$ $\sigma := S + \gamma$		
$(h_{p_1}, h_{p_2}, z_p, \alpha_g, \alpha_h, \varepsilon, \rho, \sigma)$		

Figure 6.7: Protocol P$_{OS}$ based on Okamoto-Schnorr signature.

In Figure 6.7, the resulting protocol P$_{OS}$ (OS for Okamoto-Schnorr) is presented. The modification consists of adding the values h_{w_1}, h_{w_2} and in consequence the values z_w, a_h, α_h. As in the scheme from [CMS96], the indices w and p indicate that a value is shared among $\mathcal{U}$ (who plays the part of $\mathcal{V}$ in protocol P$_{OS}$) and someone else during **w**ithdrawal or during **p**ayment; “someone else” means $\mathcal{B}$ or $\mathcal{S}$. Corresponding tuples (h_{w_1}, h_{w_2}, z_w) and (h_{p_1}, h_{p_2}, z_p) are linked by the blinding exponent ϕ.

The verification equations for the signature provided by protocol P$_{OS}$ are

$$\varepsilon \stackrel{?}{=} \mathcal{H}(m||\alpha_g||\alpha_h)$$

$$\alpha_g \stackrel{?}{=} g^{\rho} h^{\sigma} y^{\varepsilon}$$

$$\alpha_h \stackrel{?}{=} h_{p_1}^{\rho} h_{p_2}^{\sigma} z_p^{\varepsilon}.$$

If the signature results from a correct execution of the protocol, the first of them holds per definition;

the other two hold since

$$\begin{aligned}
g^{\rho}h^{\sigma}y^{\varepsilon} &= g^{t+er+\beta}h^{u+es+\gamma}y^{\varepsilon}\\
&= g^{t}h^{u}g^{\beta}h^{\gamma}(g^{r}h^{s})^{e}(g^{-r}h^{-s})^{\varepsilon}\\
&= a_{g}g^{\beta}h^{\gamma}(g^{r}h^{s})^{\varepsilon-\delta}(g^{-r}h^{-s})^{\varepsilon}\\
&= a_{g}g^{\beta}h^{\gamma}(g^{-r\delta}h^{-s\delta})\\
&= a_{g}g^{\beta}h^{\gamma}y^{\delta} = \alpha_{g}
\end{aligned}$$

$$\begin{aligned}
h_{p_1}^{\rho}h_{p_2}^{\sigma}z_{p}^{\varepsilon} &= h_{p_1}^{t+er+\beta}h_{p_2}^{u+es+\gamma}z_{p}^{\varepsilon}\\
&= h_{p_1}^{t}h_{p_2}^{u}h_{p_1}^{\beta}h_{p_2}^{\gamma}(h_{p_1}^{r}h_{p_2}^{s})^{e}(h_{p_1}^{-r}h_{p_2}^{-s})^{\varepsilon}\\
&= a_{h}^{\phi}h_{p_1}^{\beta}h_{p_2}^{\gamma}(h_{p_1}^{r}h_{p_2}^{s})^{\varepsilon-\delta}(h_{p_1}^{-r}h_{p_2}^{-s})^{\varepsilon}\\
&= a_{h}^{\phi}h_{p_1}^{\beta}h_{p_2}^{\gamma}(h_{p_1}^{-r\delta}h_{p_2}^{-s\delta})\\
&= a_{h}^{\phi}h_{p_1}^{\beta}h_{p_2}^{\gamma}z_{p}^{\delta} = \alpha_{h}.
\end{aligned}$$

Security of Protocol P_{OS}

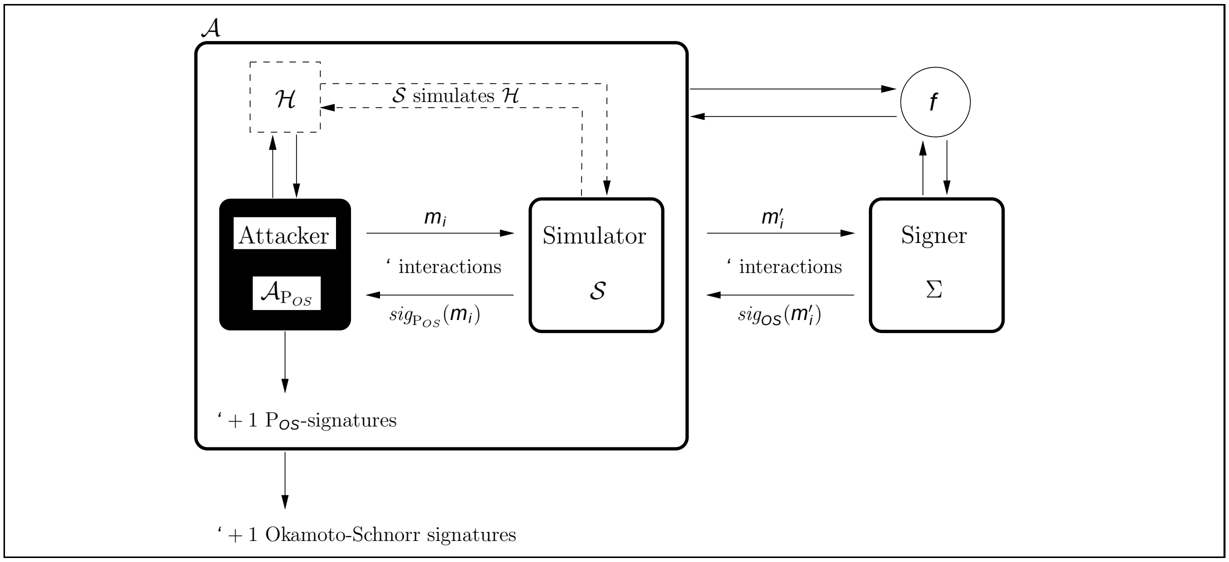

$\mathcal{A}$
$\mathcal{H}$
$\mathcal{S}$ simulates $\mathcal{H}$
f
Attacker
$\mathcal{A}_{\mathrm{P}_{OS}}$
m_i
‘ interactions
$sig_{\mathrm{P}_{OS}}(m_i)$
Simulator
$\mathcal{S}$
m_i'
‘ interactions
$sig_{OS}(m_i')$
Signer
Σ
‘ + 1 P_{OS}-signatures
‘ + 1 Okamoto-Schnorr signatures

Figure 6.8: Black box reduction for proving the security of protocol P_{OS}.

The security of our scheme against coin forgery relies on that of protocol P_{OS} against signature forgery. In the following, we'll try to prove that forging a signature after logarithmically many executions of protocol P_{OS} is as hard as doing so with the Okamoto-Schnorr signature scheme. The proof is by so-called black box reduction as presented in Figure 6.8: we assume that there is a probabilistic polynomial-time attacker $\mathcal{A}_{\mathrm{P}_{OS}}$ who can successfully obtain $\ell+1$ valid signatures after only ℓ executions of protocol P_{OS}. We use $\mathcal{A}_{\mathrm{P}_{OS}}$ to construct a probabilistic polynomial-time attacker $\mathcal{A}$ that will succeed in obtaining $\ell+1$ valid Okamoto-Schnorr signatures after only ℓ executions of the signature protocol. $\mathcal{A}_{\mathrm{P}_{OS}}$ is seen as a black box in the usual sense, i.e. as an algorithm whose input and output behavior are known, but whose interior is hidden. The description of $\mathcal{A}_{\mathrm{P}_{OS}}$ would be "an algorithm that interacts ℓ times with a signer in protocol P_{OS} and outputs $\ell+1$ valid signatures". Due to the black box character of $\mathcal{A}_{\mathrm{P}_{OS}}$, $\mathcal{A}$ has to simulate a P_{OS}-signer in an indistinguishable way to $\mathcal{A}_{\mathrm{P}_{OS}}$. $\mathcal{A}$ has control over the hash function $\mathcal{H}$ used by $\mathcal{A}_{\mathrm{P}_{OS}}$; this function has also to be simulated in such a way that $\mathcal{A}_{\mathrm{P}_{OS}}$ won't be able to tell the difference. $\mathcal{A}$ may use the Okamoto-Schnorr signer Σ as an oracle, i.e. perform regular executions of the signature protocol with him [20]. Both $\mathcal{A}$ and Σ have access to a random oracle f.

[20] Obviously not more than ‘.

Regrettably, there's no simulator $\mathcal{S}$ that would work in the desired manner. The problem is that Σ's secret key is needed to compute z_w; additionally, the random values t, u are needed to compute a_h. Nevertheless, we'll see what the simulator would look like if these problems would disappear; like this, we'll see another problem (that can be fixed), and try to do a "realistic" black box reduction for the whole payment scheme, which consists of more than just protocol P_{OS}, later on. It would clearly have been more elegant (and easier) to show that the security of the Okamoto-Schnorr scheme implies that of protocol P_{OS}.

Each of the ℓ rounds of interaction between $\mathcal{A}_{\mathrm{P}_{OS}}$ and the "magic" $\mathcal{S}$ would look like this:

- $\mathcal{S}$ engages in an execution of the Okamoto-Schnorr scheme with Σ and obtains $a := g^t h^u$ for random values $t, u \in_R \mathbf{Z}_q$. $\mathcal{S}$ sets $a_g := a$, and "magically obtains" $a_h := g^t h^u$ and $z_w := h_{w_1}^{-r} h_{w_2}^{-s}$ (magic is required since $\mathcal{S}$ doesn't know the values r, s, t, u).
- $\mathcal{S}$ sends z_w, a_g, a_h to $\mathcal{A}_{\mathrm{P}_{OS}}$, who performs all computations according to the protocol in Figure 6.7. When $\mathcal{A}_{\mathrm{P}_{OS}}$ asks $(m||\alpha_g||\alpha_h)$ to the hash function $\mathcal{H}$, $\mathcal{S}$ lets it output $f(m||\alpha_g)$ and receives $\mathcal{A}_{\mathrm{P}_{OS}}$'s challenge e.
- $\mathcal{S}$ passes e on to Σ, who responds R, S according to the Okamoto-Schnorr scheme.
- $\mathcal{S}$ forwards R, S to $\mathcal{A}_{\mathrm{P}_{OS}}$.

All the signatures output by $\mathcal{A}_{\mathrm{P}_{OS}}$ can be transformed to valid Okamoto-Schnorr signatures: for any of them consisting of $(h_{p_1}, h_{p_2}, z_p, \alpha_g, \alpha_h, \varepsilon, \rho, \sigma)$, the corresponding Okamoto-Schnorr signature is $(\alpha_g, \varepsilon, \rho, \sigma)$. The verification equations

$$\varepsilon \stackrel{?}{=} f(m||\alpha_g)$$

$$\alpha_g \stackrel{?}{=} g^\rho h^\sigma y^\varepsilon$$

hold. For the first equation, we can assume that $\mathcal{A}_{\mathrm{P}_{OS}}$ has asked $\mathcal{H}(m||\alpha_g||\alpha_h)$ for some α_h during the attack. This is because the output of $\mathcal{H}$ is seen as truly random; therefore, without asking $\mathcal{H}(m||\alpha_g||\alpha_h)$, $\mathcal{A}_{\mathrm{P}_{OS}}$ can at best guess values $\varepsilon, m, \alpha_g, \alpha_h$ such that the equation

$$\varepsilon = \mathcal{H}(m||\alpha_g||\alpha_h)$$

holds. The second equation holds because it is a verification equation for signatures resulting from protocol P_{OS} and the values in it result from the responses of the real P_{OS}-signer Σ.

Now, we have to show that $\mathcal{S}$ is indistinguishable from a real signer to $\mathcal{A}_{\mathrm{P}_{OS}}$. The value a_g sent by $\mathcal{S}$ is identical to the one a regular signer would have computed; so are the values z_w, a_h (recall our "magic assumption") and the responses R, S. The only problem is the hash function $\mathcal{H}$. It can be distinguished very easily from a real oracle, since

$$\mathcal{H}(m||\alpha_g||\alpha_h) := f(m||\alpha_g) =: \mathcal{H}(m||\alpha_g||\alpha'_h)$$

for all $m, \alpha_g, \alpha_h, \alpha'_h$. We'll now let $\mathcal{S}$ use a history table *Hist* where queries to $\mathcal{H}$ can be stored together with the response *resp* given. Then, $\mathcal{H}$ is redefined:

$$\mathcal{H}(m||\alpha_g||\alpha_h) := \begin{cases} f(m||\alpha_g) & \text{if} \quad \forall x, \mathit{resp} : (m||\alpha_g||x, \mathit{resp}) \notin \mathit{Hist} \\ \Omega \in_R \mathbf{Z}_q & \text{if} \quad \exists x, \mathit{resp} : (m||\alpha_g||x, \mathit{resp}) \in \mathit{Hist} \wedge x \neq \alpha_h \\ \mathit{resp} & \text{if} \quad (m||\alpha_g||\alpha_h, \mathit{resp}) \in \mathit{Hist.i} \end{cases}$$

Like this, the response is computed as $f(m||\alpha_g)$, if this is the first query of the form $(m||\alpha_g||x)$ for all x. If a collision occurs, i.e. if a query $(m||\alpha_g||x)$ with $x \neq \alpha_h$ has already been asked before, the response is chosen as a random value. Since $f(m||\alpha_g)$ is also truly random, in both cases the distribution of the responses is indistinguishable from a real random oracle (actually, this one is "just as random"). The third possibility is that the query has already been asked before; also in this case $\mathcal{H}$ behaves exactly like a random oracle, giving the same response as before.

Unfortunately, a new problem arises: if a collision occurs for some query $(m||\alpha_g||\alpha_h)$ and one of the $\ell + 1$ signatures that $\mathcal{A}_{\mathrm{P}_{OS}}$ outputs contains

$$\varepsilon = \mathcal{H}(m||\alpha_g||\alpha_h) := \Omega \in_R \mathbf{Z}_q,$$

then the Okamoto-Schnorr signature resulting from this signature won't be valid, since

$$\varepsilon \neq f(m||\alpha_g).$$

Assumption 6.11. Consider the black box reduction using a probabilistic polynomial-time attacker $\mathcal{A}_{\mathrm{P}_{OS}}$, a simulator $\mathcal{S}$, and a hash function $\mathcal{H}$ as described above. Then, collisions of $\mathcal{H}$ that result in invalid Okamoto-Schnorr signatures occur only with negligible probability, or $\mathcal{A}_{\mathrm{P}_{OS}}$ can be replaced by another Turing machine that performs the attack with the same probability of success and causes a collision only with negligible probability.

Argumentation. Assume that a collision occurs such that $\mathcal{A}_{\mathrm{P}_{OS}}$ asks $m||\alpha_g||\alpha'_h$ to $\mathcal{H}$ and $(m||\alpha_g||\alpha_h)$ has already been asked before. If one of these two queries is useless, i.e. if it isn't used in any way for the attack, replace $\mathcal{A}_{\mathrm{P}_{OS}}$ by another Turing machine that acts exactly like $\mathcal{A}_{\mathrm{P}_{OS}}$, with the exception that the useless query is not asked [21]. If both queries are useful, we have three cases:

1. Both queries result in a signature, i.e. among the $\ell+1$ signatures that are output, there is one which contains the values α_g, α_h, and one which contains α_g, α'_h. We can assume that the corresponding values ρ, σ and ρ', σ' differ. Since both signatures are valid, we have

   $$\alpha_g = g^{\rho} h^{\sigma} y^{f(m||\alpha_g)} = g^{\rho'} h^{\sigma'} y^{\Omega}$$

   with Ω the response to the colliding query. As both $f(m||\alpha_g)$ and Ω are truly random, with overwhelming probability there are two different representations

   $$\begin{aligned} \alpha_g &= g^{\rho - rf(m||\alpha_g)} h^{\sigma - sf(m||\alpha_g)} \\ &= g^{\rho' - r\Omega} h^{\sigma' - s\Omega}. \end{aligned}$$

   Anyone knowing the values $r, s, f(m||\alpha_g), \Omega$ may use these representations to compute the discrete logarithm of h with respect to g as

   $$\log_g h = \frac{\rho - rf(m||\alpha_g) - \rho' + r\Omega}{\sigma' - s\Omega - \sigma + sf(m||\alpha_g)}.$$

   Thus, if we let the simulator $\mathcal{S}$ choose his own secret key pair (r', s') and interact as a regular signer with $\mathcal{A}_{\mathrm{P}_{OS}}$, then with nonnegligible probability, the collusion of $\mathcal{S}$ and $\mathcal{A}_{\mathrm{P}_{OS}}$ will output $\log_g h$ in polynomial time. This is believed to be infeasible.

2. At most one of the queries results in a signature, but both are somehow useful and therefore necessary for the attack. This is improbable, since $\mathcal{H}$'s outputs are seen as truly random and thus can't be of any use unless they result in a signature. If $\mathcal{A}_{\mathrm{P}_{OS}}$ needs random values, another Turing machine exists that acts just like $\mathcal{A}_{\mathrm{P}_{OS}}$ but doesn't cause collisions (it gets the necessary random values by some other means).

3. The collision is purely casual; the probability for this is $1/q$, and therefore negligible.

The argumentation for Assumption 6.11 is not completely satisfactory; for the second case, one might imagine that $\mathcal{A}_{\mathrm{P}_{OS}}$ chooses values m, α_g and then asks several queries to $\mathcal{H}$ with different values α_h, until the response is of some specific structure which facilitates the attack. Fortunately, we can circumvent the whole problem by a slight modification to the Okamoto-Schnorr signature scheme.

Modified Blind Okamoto-Schnorr Signatures

Consider the blind Okamoto-Schnorr signature protocol as in Figure 6.3. We modify it slightly by letting $\mathcal{V}$ choose a random value α' and compute $\varepsilon := \mathcal{H}(m||\alpha||\alpha')$. The resulting protocol is shown in Figure 6.9. The value α' is part of the new signature, which is verified by checking

$$\varepsilon \overset{?}{=} \mathcal{H}(m||\alpha||\alpha')$$

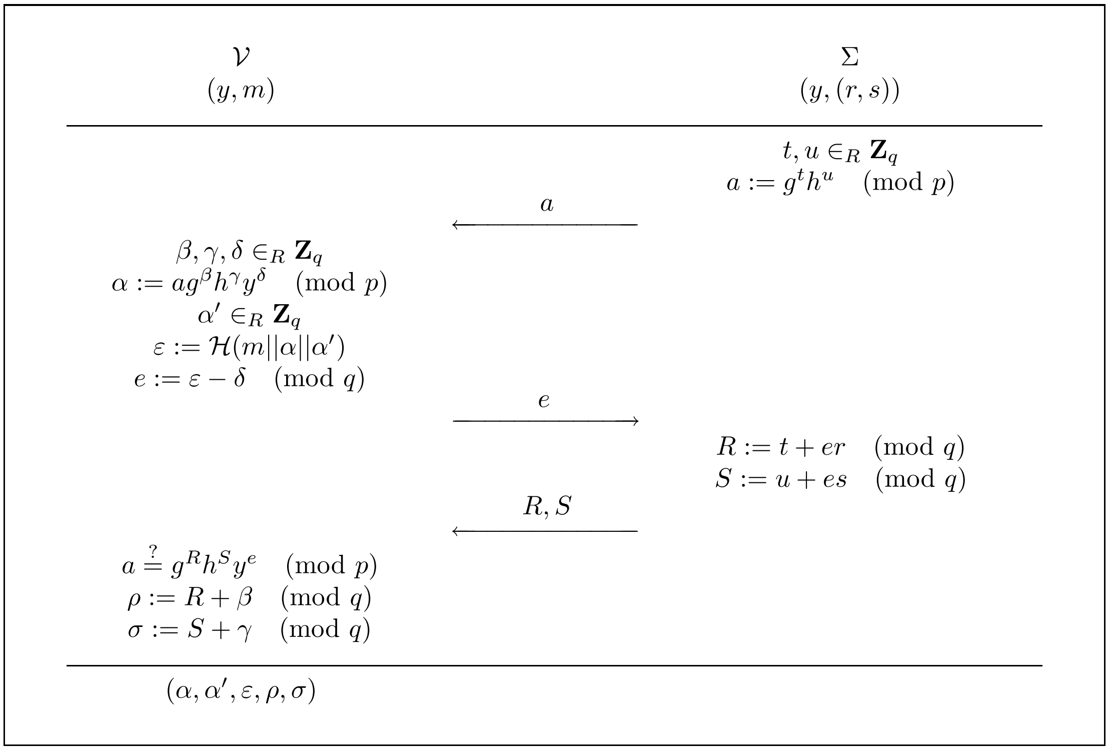

$\mathcal{V}$
(y, m)
Σ
$(y, (r, s))$
$t, u \in_R \mathbf{Z}_q$
$a := g^t h^u \pmod p$
a
$\beta, \gamma, \delta \in_R \mathbf{Z}_q$
$\alpha := a g^\beta h^\gamma y^\delta \pmod p$
$\alpha' \in_R \mathbf{Z}_q$
$\varepsilon := \mathcal{H}(m||\alpha||\alpha')$
$e := \varepsilon - \delta \pmod q$
e
$R := t + er \pmod q$
$S := u + es \pmod q$
R, S
$a \overset{?}{=} g^R h^S y^e \pmod p$
$\rho := R + \beta \pmod q$
$\sigma := S + \gamma \pmod q$
$(\alpha, \alpha', \varepsilon, \rho, \sigma)$

Figure 6.9: Modified blind Okamoto-Schnorr signature.

$$\alpha \overset{?}{=} g^\rho h^\sigma y^\varepsilon.$$

Any signature resulting from protocol P_{OS} is also a valid signature in the modified Okamoto-Schnorr scheme. The only thing that remains to show is that the modified scheme is as secure against an $(\ell, \ell+1)$-forgery as the basic Okamoto-Schnorr scheme. Intuitively, this is clear since $\mathcal{V}$ doesn't get any new information from Σ; moreover, the modifications to $\mathcal{V}$'s computations don't give him any new possibility to cheat, since $\mathcal{H}$ is seen as a random oracle. Formally, we have to investigate whether the forking lemma for blind signatures is still valid. This is the case, as can be easily verified (we only have to insert α' in the appropriate positions of the proof). Analogously, Theorem 6.6 holds also for the modified Okamoto-Schnorr scheme.

With this result, we implement the simulator $\mathcal{S}$ as described before, but let the response to the query $m||\alpha_g||\alpha_h$ be $f(m||\alpha_g||\alpha_h)$ instead of $f(m||\alpha_g)$. Under the "magic assumption", $\mathcal{A}_{\mathrm{P}_{OS}}$ can't distinguish $\mathcal{S}$ from a modified Okamoto-Schnorr signer.

Now, back to reality and to the problem that still persists: we can't construct a simulator $\mathcal{S}$ for protocol P_{OS} as above (because our "magic assumption" doesn't hold and there's no way to simulate the signer without the values r, s, t, u), and therefore we'll build the complete payment system and then try to prove its security against coin forgery.

Adding a Checker to Protocol P_{OS}

Figure 6.10 shows protocol P_{OSC}, which is the result of adding a checker to protocol P_{OS}. The signer Σ performs the tasks of both the signer from protocol P_{OS} and the checker. Moreover, we use additional hash functions $\mathcal{H}_0, \mathcal{H}_1$ to compute the values μ_i, c_i; like this we don't have to worry about whether $\mathcal{V}$

[21] We have to allow $\mathcal{A}_{\mathrm{P}_{OS}}$ to perform useless computations, since we don't know anything about the machine's interior. But obviously if a "stupid" machine exists, there's also an "intelligent" version of it. This is the positive aspect of the black box character: not only can't we say anything about how the machine actually looks like, we also don't need to do so.

$\mathcal{V}$		Σ
$(y, m, h_{w_1}, h_{w_2}, \phi)$		$(y, (r, s), h_{w_1}, h_{w_2})$
for $i \in \{0, 1\}$: $\beta_i, \gamma_i, \delta_i, \varphi_i, \nu_i \in_R \mathbf{Z}_q$ $\mu_i := \mathcal{H}_0(m \vert\vert \varphi_i)$ $c_i := \mathcal{H}_1(\beta_i \vert\vert \gamma_i \vert\vert \delta_i \vert\vert \mu_i \vert\vert \nu_i)$		
	$\xrightarrow{c_0, c_1}$	
		$z_w := h_{w_1}^{-r} h_{w_2}^{-s}$ for $i \in \{0, 1\}$: $t_i, u_i \in_R \mathbf{Z}_q$ $a_{g,i} := g^{t_i} h^{u_i}$ $a_{h,i} := h_{w_1}^{t_i} h_{w_2}^{u_i}$
	$\xleftarrow{z_w, a_{g,0/1}, a_{h,0/1}}$	
$h_{p_1} := h_{w_1}^{\phi}$ $h_{p_2} := h_{w_2}^{\phi}$ $z_p := z_w^{\phi}$ for $i \in \{0, 1\}$: $\alpha_{g,i} := a_{g,i} g^{\beta_i} h^{\gamma_i} y^{\delta_i}$ $\alpha_{h,i} := a_{h,i}^{\phi} h_{p_1}^{\beta_i} h_{p_2}^{\gamma_i} z_p^{\delta_i}$ $\varepsilon_i := \mathcal{H}(\mu_i \vert\vert \alpha_{g,i} \vert\vert \alpha_{h,i})$ $e_i := \varepsilon_i - \delta_i$		
	$\xrightarrow{e_0, e_1}$	
		$I \in_R \{0, 1\}$
	$\xleftarrow{I}$	
	$\xrightarrow{\beta_I, \gamma_I, \delta_I, \mu_I, \nu_I, \alpha_{h,I}}$	
		verify c_I and e_I $R := t_{\bar{I}} + e_{\bar{I}} r$ $S := u_{\bar{I}} + e_{\bar{I}} s$
	$\xleftarrow{R, S}$	
$a_{g,\bar{I}} \stackrel{?}{=} g^R h^S y^{e_{\bar{I}}}$ $a_{h,\bar{I}} \stackrel{?}{=} h_{w_1}^R h_{w_2}^S z_w^{e_{\bar{I}}}$ $\rho := R + \beta_{\bar{I}}$ $\sigma := S + \gamma_{\bar{I}}$ $\alpha_g := \alpha_{g,\bar{I}}$ $\alpha_h := \alpha_{h,\bar{I}}$ $\varepsilon := \varepsilon_{\bar{I}}$ $\mu := \mu_{\bar{I}}$		
$(\mu, h_{p_1}, h_{p_2}, z_p, \alpha_g, \alpha_h, \varepsilon, \rho, \sigma)$		

Figure 6.10: Protocol P$_{OSC}$ based on Okamoto-Schnorr signature.

gets additional information on the hash function $\mathcal{H}$. The value c_I is verified by checking

$$c_I \overset{?}{=} \mathcal{H}(\beta_I||\gamma_I||\delta_I||\mu_I||\nu_I).$$

Recall that ν_i "hides" c_i. Since the value $\alpha_{h,I}$ is also used for the computation of e_I, the verification of e_I is performed as

$$e_I \overset{?}{=} \mathcal{H}(\mu_I||a_I g^{\beta_I} h^{\gamma_I} y^{\delta_I}||\alpha_{h,I}) - \delta_I.$$

We can't let Σ compute $\alpha_{h,I}$ by himself (like $\alpha_{g,I} := a_I g^{\beta_I} h^{\gamma_I} y^{\delta_I}$), because $\mathcal{V}$ would then have to show his secret ϕ. Note that if Σ could compute discrete logarithms, he could at this time extract ϕ: with $b := a_{h,I} h_{w_1}^{\beta_I} h_{w_2}^{\gamma_I} z_w^{\delta_I}$, we have

$$\log_b \alpha_{h,I} = \phi.$$

The verification equations for the signature $(h_{p_1}, h_{p_2}, z_p, \alpha_g, \alpha_h, \varepsilon, \rho, \sigma)$ on $\mu_{\bar{I}} := \mathcal{H}(m||\varphi_{\bar{I}})$ provided by protocol P_{OSC} are

$$\varepsilon \overset{?}{=} \mathcal{H}(\mu_{\bar{I}}||\alpha_g||\alpha_h)$$
$$\alpha_g \overset{?}{=} g^{\rho} h^{\sigma} y^{\varepsilon}$$
$$\alpha_h \overset{?}{=} h_{p_1}^{\rho} h_{p_2}^{\sigma} z_p^{\varepsilon}.$$

They hold if the signature results from a correct execution of the protocol, analogously to the computations made before for protocol P_{OS}. We'll proceed with the description of the whole e-cash scheme. Protocol P_{OSC} will be used as a building block for the withdrawal protocol.

System Setup

As in Subsection 6.2.2, let G be a finite cyclic group of prime order q with generators g, h such that computing discrete logarithms is infeasible. $g_1, \ldots, g_4$ are additional generators; the participating parties are assumed not to know how to represent any of the generators with respect to the others. To achieve this, the generators should be chosen in a publicly verifiable, pseudorandom manner. Then virtually anybody might choose them; typically, this would be done either by $\mathcal{B}$ or by some governmental entity.

$\mathcal{B}$ chooses a secret key $(r, s) \in (\mathbf{Z}_q \times \mathbf{Z}_q)$; the corresponding public key is $y := g^{-r} h^{-s}$.

$\mathcal{T}$ chooses a secret key $\tau \in_R \mathbf{Z}_q^*$; the corresponding public key is $y_{\mathcal{T}} := g_2^{\tau}$.

Let $\mathcal{H}$ be a collision-resistant one-way hash function that maps $\{0,1\}^*$ to $\mathbf{Z}_q$.

Withdrawal

The withdrawal protocol is shown in 6.11. During withdrawal, $\mathcal{U}$ chooses a random coin number $c\#$ and the blinding factor $\phi \in_R \mathbf{Z}_q^*$. The "coin bases" h_{w_1}, h_{w_2} and the tracing information d are computed with ϕ. Again, d can be seen as a Diffie-Hellman-type encryption of ϕ. The proofs of equality of logarithms U_1, U_2 prove the following:

$$U_1: \quad \log_{g_1}(h_{w_1}/g_2) = \log_d y_{\mathcal{T}}$$
$$U_2: \quad \log_{g_3}(h_{w_2}/g_4) = \log_d y_{\mathcal{T}}.$$

Thus, they show that the same value ϕ was used for the computation of h_{w_1}, h_{w_2}, and d. Additionally, the correct form of these values is proven.

Upon successful verification of U_1 and U_2 by $\mathcal{B}$, both parties engage in an execution of the protocol P_{OSC} with the initial values as shown in Figure 6.11. $\mathcal{U}$ ends up with $(\mu, h_{p_1}, h_{p_2}, z_p, \alpha_g, \alpha_h, \varepsilon, \rho, \sigma)$; the coin will be

$$coin := (\mu, h_{p_1}, h_{p_2}, z_p, \alpha_g, \alpha_h, \varepsilon, \rho, \sigma, V)$$

and consist of the commitment $\mu := \mathcal{H}(c\#||\varphi_{\bar{I}})$ to the coin number $c\#$ [22], the modified Okamoto-Schnorr signature $(h_{p_1}, h_{p_2}, z_p, \varepsilon, \alpha_g, \alpha_h, \rho, \sigma)$ on μ, and a proof V about the structure of h_{p_1}, h_{p_2}. As in the system

[22] Note that as in the on-line scheme from [CMS96], this coin number isn't actually used for anything; in the off-line system presented in the next section it will be replaced by a value that will allow $\mathcal{B}$ to trace double-spenders without the help of $\mathcal{T}$.

$$\begin{array}{lcc}
\mathcal{U} & & \mathcal{B} \\
\hline
\text{random coin number } c\# & & \\
\phi \in_R \mathbf{Z}_q^* & & \\
h_{w_1} := g_1^{1/\phi} g_2 & & \\
h_{w_2} := g_3^{1/\phi} g_4 & & \\
d := y_{\mathcal{T}}^{\phi} & & \\
U_1 := \mathit{Proof}_{EQLOG}(\epsilon, g_1, (h_{w_1}/g_2), d, y_{\mathcal{T}}) & & \\
U_2 := \mathit{Proof}_{EQLOG}(\epsilon, g_3, (h_{w_2}/g_4), d, y_{\mathcal{T}}) & & \\
 & \xrightarrow{h_{w_1}, h_{w_2}, d, U_1, U_2} & \\
 & & \text{verify } U_1, U_2 \\
(y, c\#, h_{w_1}, h_{w_2}, \phi) & & (y, (r,s), h_{w_1}, h_{w_2}) \\
\hline
 & \text{protocol } \mathrm{P}_{OSC} & \\
\hline
(\mu, h_{p_1}, h_{p_2}, z_p, \alpha_g, \alpha_h, \varepsilon, \rho, \sigma) & & (d)
\end{array}$$

Figure 6.11: Withdrawal in our on-line escrow-based system.

from [CMS96], the proof V prevents the attack of choosing $h_{p_1} := h_{w_1}^{\phi} g^{\zeta}$, $h_{p_2} := h_{w_2}^{\phi} h^{\zeta}$, and $z_p := z_w^{\phi} y^{\zeta}$ for some $\zeta \neq 0$. Such an attack would yield a valid untraceable coin.

V is defined as

$$V := \mathit{Proof}_{EQLOG}(\epsilon, g_2, h_{p_1}/g_1, g_4, h_{p_2}/g_3)$$

and proves that

$$\log_{g_2}(h_{p_1}/g_1) = \log_{g_4}(h_{p_2}/g_3).$$

If implemented in the same way as the proof from [CMS96] described in Subsection 2.7.4, it consists of a pair (c_V, s_V) and is verified by checking

$$c_V \overset{?}{=} \mathcal{H}(\epsilon || g_2 || h_{p_1}/g_1 || g_4 || h_{p_2}/g_3 || g_2^{s_V} (h_{p_1}/g_1)^{c_V} || g_4^{s_V} (h_{p_2}/g_3)^{c_V}).$$

$\mathcal{B}$ obtains the encrypted tracing information d which is stored in the withdrawal database.

Payment

The payment protocol is shown in Figure 6.12; $\mathcal{U}$ computes V (this might of course also be done before payment) and sends the coin to $\mathcal{S}$, who verifies the modified Okamoto-Schnorr signature and the proof V. The coin is then passed on to $\mathcal{B}$, who in addition checks in his deposit database that *coin* hasn't already been spent.

Anonymity Revocation

The anonymity can be revoked by $\mathcal{T}$ by use of his secret key τ; the two different kinds of anonymity revocation are performed as follows:

- Payment-based (owner tracing): $\mathcal{T}$ is given an h_{p_1} observed in a payment; $\mathcal{T}$ computes $(h_{p_1}/g_1)^{\tau} = (g_2^{\phi})^{\tau} = d$, and $\mathcal{B}$ searches his withdrawal database for d.
- Withdrawal-based (coin tracing): $\mathcal{T}$ is given a d observed in a withdrawal; $\mathcal{T}$ computes $g_1 d^{1/\tau} = g_1 g_2^{\phi} = h_{p_1}$, and h_{p_1} can be put on a blacklist for recognizing the coin when it is spent.

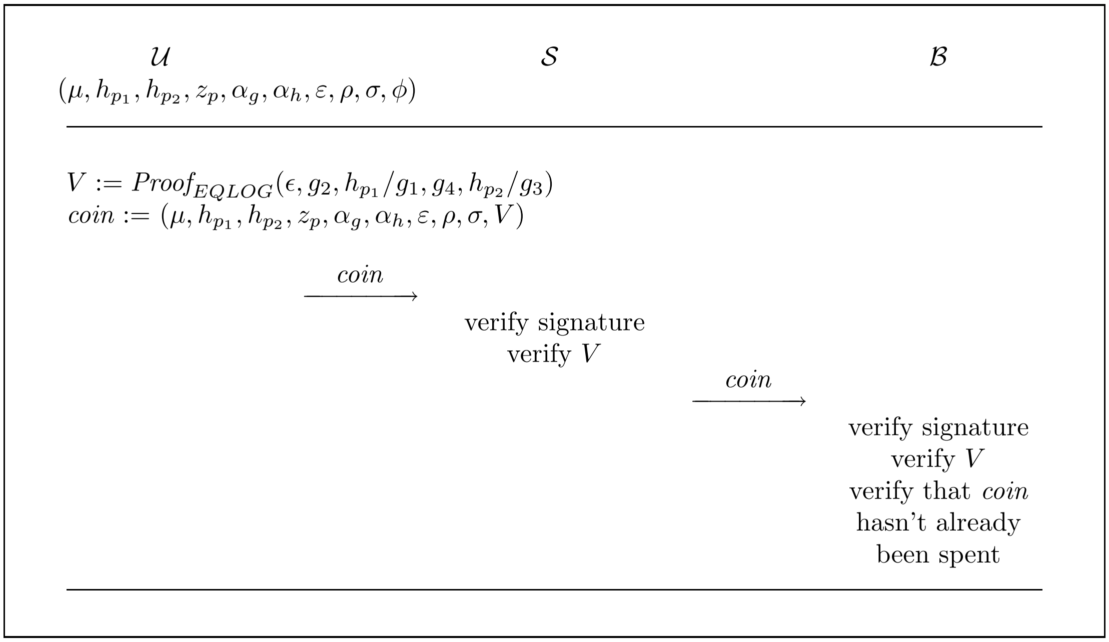

$\mathcal{U}$
$\mathcal{S}$
$\mathcal{B}$
$(\mu, h_{p_1}, h_{p_2}, z_p, \alpha_g, \alpha_h, \varepsilon, \rho, \sigma, \phi)$
$V := \mathit{Proof}_{EQLOG}(\epsilon, g_2, h_{p_1}/g_1, g_4, h_{p_2}/g_3)$
$\mathit{coin} := (\mu, h_{p_1}, h_{p_2}, z_p, \alpha_g, \alpha_h, \varepsilon, \rho, \sigma, V)$
$\xrightarrow{\mathit{coin}}$
verify signature
verify V
$\xrightarrow{\mathit{coin}}$
verify signature
verify V
verify that *coin*
hasn't already
been spent

Figure 6.12: Payment in our on-line escrow-based system.

The equalities used above hold because for every triple (h_{w_1}, h_{p_1}, d) originating from a correct withdrawal it is true that

$$\phi = (\log_{g_1}(h_{w_1}/g_2))^{-1} = \log_{y_\mathcal{T}} d = \log_{g_2}(h_{p_1}/g_1).$$

6.4.2 Security of the Scheme

Security for $\mathcal{U}$

Our arguments on the security for $\mathcal{U}$ are similar to those for the on-line system from [CMS96] in Subsection 4.3.1. This is a direct consequence of the similarity of the underlying blind signature schemes.

For payments to be unlinkable and anonymous, the protocol P_{OSC} has to be blind; unconditional blindness could be proven by showing that for every possible view of $\mathcal{B}$ and for every possible message-signature tuple there's exactly one suitable tuple $(\phi, \beta_i, \gamma_i, \delta_i, \varphi_i, \nu_i)$ of blinding factors. Regrettably, protocol P_{OSC} isn't unconditionally blind; this is due to the fact that $\mathcal{V}$ has to send $\alpha_{h,I}$ to Σ. As we stated before, the secret value ϕ could then be computed as the discrete logarithm of $\alpha_{h,I}$ to an appropriate base. Hence, the ability to compute discrete logarithms would break the blindness of protocol P_{OSC}. If we can show that this is the only way to break it, we may still be satisfied; the anonymity of the whole scheme is based on the stronger Decision Diffie-Hellman assumption, and therefore we wouldn't lose anything.

Let's for a moment suppose $\mathcal{V}$ doesn't give the value $\alpha_{h,I}$ to Σ. Then, given a view consisting of h_{w_1}, h_{w_2}, c_i, z_w, $a_{g,i}$, $a_{h,i}$, e_i, I, β_I, γ_I, δ_I, μ_I, R, S and a message-signature tuple $(\mu_{\bar{I}}, h_{p_1}, h_{p_1}, z_p, \alpha_g, \alpha_h, \varepsilon, \rho, \sigma)$, we can say the following about the possible blinding factors:

The values β_I, γ_I, δ_I, φ_I, ν_I are clearly fixed (φ_I is fixed by μ_I, i.e. by $\mu_I := \mathcal{H}(c\#||\varphi_I)$), because $\mathcal{U}$ sends them to $\mathcal{B}$. For the remaining values, the only possibility is

$$\begin{aligned}
\phi &:= \log_{h_{w_1}} h_{p_1} = \log_{h_{w_2}} h_{p_2} = \log_{z_w} z_p \\
\beta_{\bar{I}} &:= \rho - R \\
\gamma_{\bar{I}} &:= \sigma - S \\
\delta_{\bar{I}} &:= \varepsilon_{\bar{I}} - e_{\bar{I}} \\
\varphi_{\bar{I}} &:= \text{the value such that } \mu_{\bar{I}} = \mathcal{H}(m||\varphi_{\bar{I}}).
\end{aligned}$$

With these blinding factors, $\mathcal{U}$ would have computed (for $i \in \{0, 1\}$)

$$\begin{array}{rcl}
\mu_{\bar{I}} & := & \mathcal{H}(m||\varphi_{\bar{I}}) \\
c_i & := & \mathcal{H}(\beta_i||\gamma_i||\delta_i||\mu_i||\nu_i) \\
h_{p_1} & := & h_{w_1}^{\phi} \\
h_{p_2} & := & h_{w_2}^{\phi} \\
z_p & := & z_w^{\phi} \\
\alpha_{g,i} & := & a_{g,i} g^{\beta_i} h^{\gamma_i} y^{\delta_i} \\
\alpha_{h,i} & := & a_{h,i}^{\phi} h_{p_1}^{\beta_i} h_{p_2}^{\gamma_i} z_p^{\delta_i} \\
\varepsilon_i & := & \mathcal{H}(\mu_i||\alpha_{g,i}||\alpha_{h,i}) \\
e_i & := & \varepsilon_i - \delta_i.
\end{array}$$

The verification of c_I then obviously holds; we omit the verification of e_I, because Σ can't perform it without the value $\alpha_{h,I}$. It remains to show that $\alpha_{g,\bar{I}} = g^{\rho} h^{\sigma} y^{\varepsilon_{\bar{I}}}$ and $\alpha_{h,\bar{I}} = h_{p_1}^{\rho} h_{p_2}^{\sigma} z_p^{\varepsilon_{\bar{I}}}$:

$$\alpha_{g,\bar{I}} = a_{g,\bar{I}} g^{\beta_{\bar{I}}} h^{\gamma_{\bar{I}}} y^{\delta_{\bar{I}}} = g^{t_{\bar{I}}+\beta_{\bar{I}}+re_{\bar{I}}} h^{u_{\bar{I}}+\gamma_{\bar{I}}+se_{\bar{I}}} y^{\varepsilon_{\bar{I}}} = g^{R+\beta_{\bar{I}}} h^{S+\gamma_{\bar{I}}} y^{\varepsilon_{\bar{I}}} = g^{\rho} h^{\sigma} y^{\varepsilon_{\bar{I}}}$$

$$\alpha_{h,\bar{I}} = a_{h\bar{I}}^{\phi} h_{p_1}^{\beta_{\bar{I}}} h_{p_2}^{\gamma_{\bar{I}}} z_p^{\delta_{\bar{I}}} = h_{p_1}^{t_{\bar{I}}+\beta_{\bar{I}}+re_{\bar{I}}} h_{p_2}^{u_{\bar{I}}+\gamma_{\bar{I}}+se_{\bar{I}}} z_p^{\varepsilon_{\bar{I}}} = h_{p_1}^{R+\beta_{\bar{I}}} h_{p_2}^{S+\gamma_{\bar{I}}} z_p^{\varepsilon_{\bar{I}}} = h_{p_1}^{\rho} h_{p_2}^{\sigma} z_p^{\varepsilon_{\bar{I}}}.$$

The equation $\varepsilon_{\bar{I}} := \mathcal{H}(\mu_{\bar{I}}||\alpha_{g,\bar{I}}||\alpha_{h,\bar{I}})$ holds per definition. Thus, the blinding factors $(\phi, \beta_i, \gamma_i, \delta_i, \varphi_i)$ would in fact have resulted in the valid message-signature tuple $(\mu_{\bar{I}}, h_{p_1}, h_{p_2}, z_p, \alpha_{g,\bar{I}}, \alpha_{h,\bar{I}}, \varepsilon_{\bar{I}}, \rho, \sigma)$.

This convinces us that if $\mathcal{V}$ didn't give the value $\alpha_{h,I}$ to Σ, the protocol would be unconditionally blind. Given the value $\alpha_{h,I}$ alone, Σ would have to be able to compute discrete logarithms in order to break the blindness of the protocol. We now have to ask ourselves if $\alpha_{h,I}$ together with the rest of the view could be used to break the blindness in some other way (one which is less powerful than computing discrete logarithms). This seems infeasible, since $\alpha_{h,I}$ is "thrown away" immediately after the verification of e_I, i.e. it isn't used for anything else. Furthermore, no value from the resulting signature is linked to $\alpha_{h,I}$ by anything else than ϕ, because the signature contains only blinding factors with index $\bar{I}$. Since these factors remain unknown to Σ, he can't establish a link between $\alpha_{h,I}$ and the signature.

Even if protocol P_{OSC} was unconditionally blind, $\mathcal{U}$'s anonymity would still be only computational, because the encrypted tracing information d could be used by $\mathcal{B}$ to link withdrawal and payment by testing

$$\log_{y_{\mathcal{T}}} d \overset{?}{=} \log_{g_2}(h_{p_1}/g_1).$$

This would be feasible if one could efficiently compute discrete logarithms. Being able to solve the Decision Diffie-Hellman problem would also be sufficient, the instance to solve would be

$$\text{given } (h_{p_1}/g_1) = g_2^{\phi} \text{ and } y_{\mathcal{T}} = g_2^{\tau}, \text{ decide } d \overset{?}{=} g_2^{\tau\phi}.$$

In [Cam98] it was stated that linking like this is even equivalent to the Decision Diffie-Hellman problem, but we can't see this.

The proof V from payment can't be linked to the corresponding withdrawal. Furthermore, above we stated that $\mathcal{B}$'s view from protocol P_{OSC} and the value d obtained during withdrawal both for themselves don't compromise user anonymity, at least against a computationally limited $\mathcal{B}$. What happens if we allow simultaneous evaluation of both pieces of information and possibly also of the proofs U_1, U_2? The value d is defined as the public key of the trustee taken to the power of $\mathcal{U}$'s secret value ϕ. As long as we assume that solving the discrete logarithm or the Decision Diffie-Hellman problem is infeasible, d shouldn't compromise $\mathcal{U}$'s anonymity. The same holds for the proofs U_1, U_2, which are assumed not to leak information about ϕ under the mentioned assumptions.

A rigorous proof of the infeasibility to link withdrawals to payments should try to make a reduction from some problem assumed to be hard to linking. Due to lack of time we weren't able to do this here; we suspect the hard problem to be the "hard Decision Diffie-Hellman problem" [23].

As double-spending is impossible in an on-line system, false accuses of overspending and further overspending by other parties are no issue.

[23] It is almost identical to the Decision Diffie-Hellman problem: given values g, g^x, g^y, $g^{x^{-1}}$, r, decide $r \overset{?}{=} g^{xy}$.

Security for $\mathcal{B}$

In the protocol P_{OS}, the only additional information (with respect to the basic Okamoto-Schnorr signature scheme) given by $\mathcal{B}$ to $\mathcal{U}$ is the values z_w and a_h. The question is to what extent this information actually diminishes the security against coin forgery.

Intuitively, we might argue that in protocol P_{OS}, $\mathcal{B}$'s secret values r, s, t, u are perfectly hidden inside z_w and a_h. The same holds for the secret values r, s, t_i, u_i and the additional values $z_w, a_{h,i}$ in protocol P_{OSC} (for $i \in \{0, 1\}$).

Based on this and recalling that the verification equations for signatures from protocol P_{OSC} are

$$\varepsilon \overset{?}{=} \mathcal{H}(\mu_{\bar{I}}||\alpha_g||\alpha_h)$$

$$\alpha_g \overset{?}{=} g^{\rho} h^{\sigma} y^{\varepsilon}$$

$$\alpha_h \overset{?}{=} h_{p_1}^{\rho} h_{p_2}^{\sigma} z_p^{\varepsilon},$$

we could assume that the successful forgery of a coin requires the attacker to be able to do one of the following:

- find a collision for the hash function $\mathcal{H}$
- solve the verification equations for arbitrary values
- if not able to solve the verification equations in a universal way: find, possibly using authentic signatures issued by the signer, a new, valid coin.

The hash function is seen as a random oracle and therefore finding a collision for it it infeasible.

If the attacker was able to universally solve the verification equations, forging Okamoto-Schnorr signatures (even with the transformation from [Poi98]) would be equally possible for him.

What remains is the third possibility; it means finding a coin that hasn't been issued by $\mathcal{B}$ and that can't be recognized as a forgery. Hence, the values μ, h_{p_1}, h_{p_2}, and z_p of this new coin must be different from those of previously withdrawn coins; otherwise $\mathcal{B}$ would detect the coin when checking for double-spending in the deposit database. It seems reasonable that if the attacker was able to find corresponding values $\alpha_g, \alpha_h, \varepsilon, \rho, \sigma$ such that the verification equations hold for the resulting signature, he could do so also in the case of Okamoto-Schnorr signatures with checker. Hence, a probabilistic polynomial-time attacker capable of forging coins in this payment system could also forge signatures in the signature scheme from [Poi98].

The argumentation above seems plausible, but it doesn't strictly prove anything (because we don't know how much more information $\mathcal{A}$ actually gets from z_w, $a_{h,i}$). This is why we now return to black box reduction. Let's assume that there is a probabilistic polynomial-time attacker $\mathcal{A}_{\mathrm{P}_{OSC}}$ for our payment system who succeeds, with nonnegligible probability, in an $(\ell, \ell+1)$-forgery. Now, we'd like to use $\mathcal{A}_{\mathrm{P}_{OSC}}$ as a black box to build a probabilistic polynomial-time attacker $\mathcal{A}$ that performs, with nonnegligible probability, an $(\ell, \ell+1)$-forgery in the modified Okamoto-Schnorr scheme with checker. The setting would then be as in Figure 6.13.

The question that immediately arises is: can we construct the simulator $\mathcal{S}$ or do we have the same problem as before, i.e. do we need the signer's secret key?

Clearly, $\mathcal{A}_{\mathrm{P}_{OSC}}$ will probably verify that the responses R, S obtained by him satisfy the verifications from protocol P_{OSC}

$$a_{g,\bar{I}} \overset{?}{=} g^R h^S y^{e_{\bar{I}}},$$

$$a_{h,\bar{I}} \overset{?}{=} h_{w_1}^R h_{w_2}^S z_w^{e_{\bar{I}}}.$$

We'll have to circumvent the problem that the signer's secrets (r, s, t_i, u_i) are still needed to compute z_w and $a_{h,i}$. If h_{w_1} and h_{w_2} were of the special form $h_{w_1} = g^{\eta}$ and $h_{w_2} = h^{\eta}$ for a value $\eta \in \mathbf{Z}_q$ known to $\mathcal{S}$, he could compute

$$z_w := y^{\eta} = g^{-r\eta} h^{-s\eta} = h_{w_1}^{-r} h_{w_2}^{-s},$$

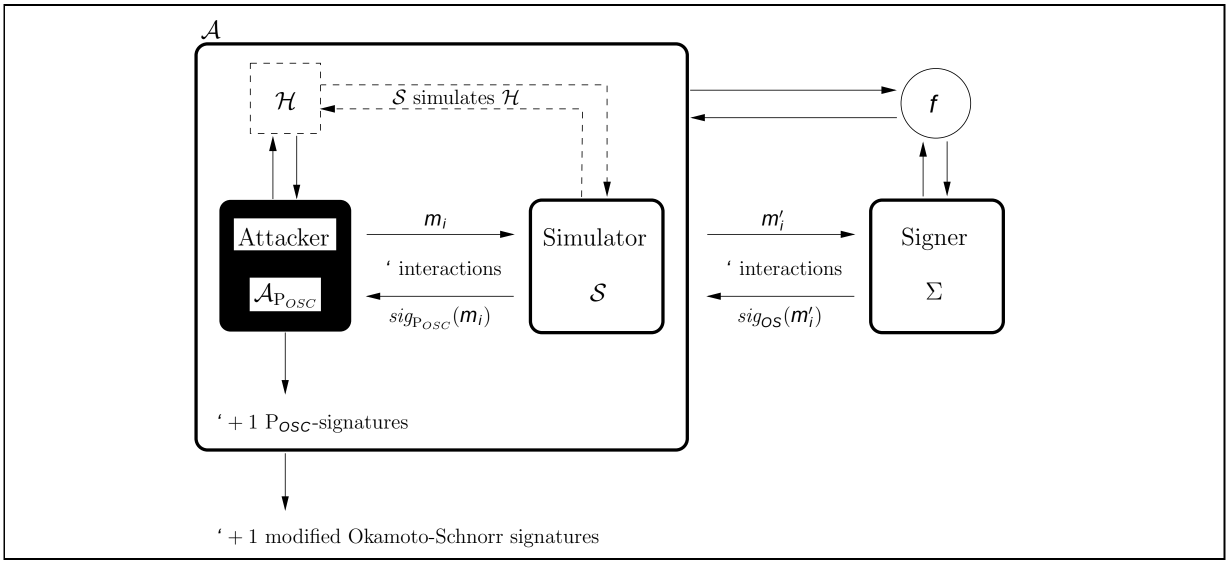

$\mathcal{A}$
$\mathcal{H}$
$\mathcal{S}$ simulates $\mathcal{H}$
f
m_i
Attacker
$\mathcal{A}_{\mathrm{P}_{OSC}}$
‘ interactions
$sig_{\mathrm{P}_{OSC}}(m_i)$
Simulator
$\mathcal{S}$
m_i'
‘ interactions
$sig_{OS}(m_i')$
Signer
Σ
‘ + 1 P_{OSC}-signatures
‘ + 1 modified Okamoto-Schnorr signatures

Figure 6.13: Black box reduction for proving the security of our e-cash system.

$$a_{h,i} := a_{g,i}^{\eta} = g^{t_i \eta} h^{u_i \eta} = h_{w_1}^{t_i} h_{w_2}^{u_i},$$

and z_w, $a_{h,i}$ would thus have the correct form. $\mathcal{S}$ can achieve this if he can get to know ϕ and if he can choose the generators $g_1, \ldots, g_4$. Assuming that $\mathcal{S}$ obtains ϕ, he'll choose $g_1, \ldots, g_4$ as follows:

1. choose $b_1, b_2 \in_R \mathbf{Z}_q$,
2. set $g_1 := g^{b_1}, g_2 := g^{b_2}$ and $g_3 := h^{b_1}, g_4 := h^{b_2}$.

Then, with η computed as $\eta := b_1/\phi + b_2$, we have

$$g^{\eta} = g^{b_1/\phi + b_2} = g_1^{1/\phi} g_2 = h_{w_1},$$

$$h^{\eta} = h^{b_1/\phi + b_2} = g_3^{1/\phi} g_4 = h_{w_2}.$$

Thus, $\mathcal{S}$ can provide $\mathcal{A}_{\mathrm{P}_{OSC}}$ with correct values z_w, $a_{h,i}$. We'll have to address the following questions:

- How does $\mathcal{S}$ obtain ϕ?
- Does it change the probability of success of $\mathcal{A}_{\mathrm{P}_{OSC}}$ if we let $\mathcal{S}$ choose $g_1, \ldots, g_4$, i.e. do these generators have the same probability distribution as if they were chosen randomly?
- Is $\mathcal{S}$ still indistinguishable for $\mathcal{A}_{\mathrm{P}_{OSC}}$?

We'll let $\mathcal{S}$ "extract" ϕ; to this end, the proofs U_1, U_2 in the withdrawal protocol (Figure 6.11) are made interactive. Actually, it suffices to make one of them interactive; we show this in Figure 6.14 for U_1.

Now, we let $\mathcal{S}$ execute the interactive version of U_1 once; he obtains the response r. Then he resets $\mathcal{A}_{\mathrm{P}_{OSC}}$ to the point where $\mathcal{A}_{\mathrm{P}_{OSC}}$ has just sent the commitments a, b. $\mathcal{S}$ then sends another challenge $c' \neq c$ and obtains another response r'. He can then compute ϕ as

$$\phi = \frac{c - c'}{r - r'}.$$

The resetting of the attacker doesn't contradict his black box character, since we don't modify the interior of $\mathcal{A}_{\mathrm{P}_{OSC}}$ nor need any knowledge about it. Skeptical readers may think of it like this: instead of resetting $\mathcal{A}_{\mathrm{P}_{OSC}}$, we may as well run two machines, both identical to $\mathcal{A}_{\mathrm{P}_{OSC}}$, with the same random tapes and inputs, with exception of the challenges: c for one machine, c' for the other. Having received r and r', we switch one of the machines off.

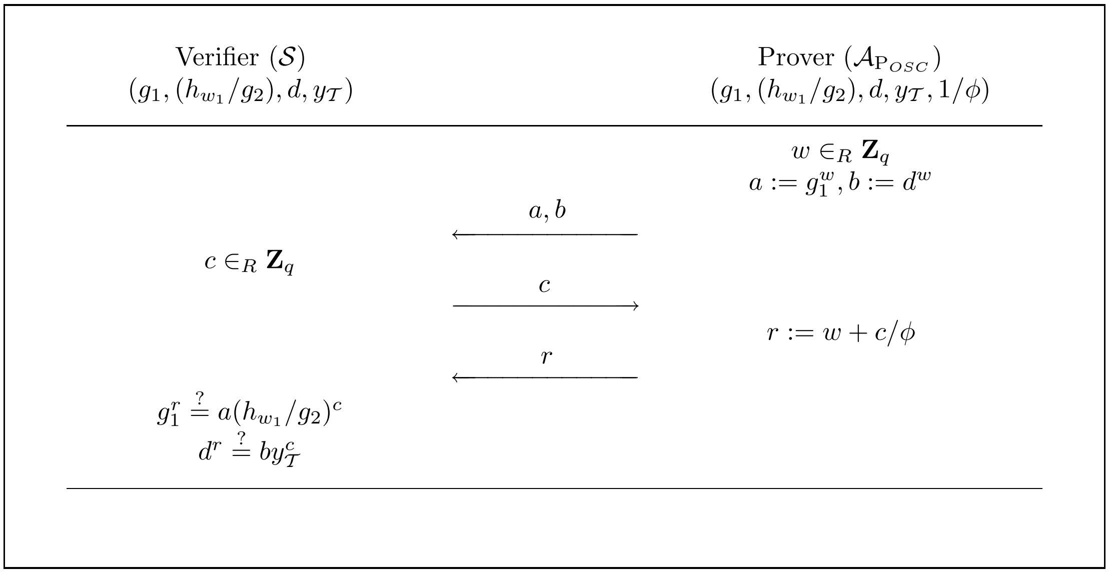

Verifier ($\mathcal{S}$)
$(g_1, (h_{w_1}/g_2), d, y_{\mathcal{T}})$
Prover ($\mathcal{A}_{\mathrm{P}_{OSC}}$)
$(g_1, (h_{w_1}/g_2), d, y_{\mathcal{T}}, 1/\phi)$
$w \in_R \mathbf{Z}_q$
$a := g_1^w, b := d^w$
a, b
$c \in_R \mathbf{Z}_q$
c
$r := w + c/\phi$
r
$g_1^r \overset{?}{=} a(h_{w_1}/g_2)^c$
$d^r \overset{?}{=} b y_{\mathcal{T}}^c$

Figure 6.14: Interactive version of the proof U_1.

Next, we'll see whether the generators $g_1, \ldots, g_4$ chosen by $\mathcal{S}$ as described above will have a different probability distribution than generators chosen at random. The generators $g_1, \ldots, g_4$ are based on the randomly chosen generators g, h of the Okamoto-Schnorr signature. Since the values b_1, b_2 we use to compute $g_1, \ldots, g_4$ are also chosen randomly, the resulting generators are "almost" random. They are related to each other by the equations

$$\log_g g_1 = \log_h g_3 \quad \text{and} \quad \log_g g_2 = \log_h g_4.$$

But we can assume that $\mathcal{A}_{\mathrm{P}_{OSC}}$ can't distinguish tuples $g_1, \ldots, g_4$ for which these relations hold from tuples for which they don't hold [24]. Furthermore, these relations shouldn't diminish the success probability of $\mathcal{A}_{\mathrm{P}_{OSC}}$; and even if they did so, they would have an impact only on the third verification equation

$$\alpha_h \overset{?}{=} h_{p_1}^{\rho} h_{p_2}^{\sigma} z_p^{\varepsilon}.$$

Hence, the $\ell+1$ P_{OSC}-signatures might not all be valid, but the resulting $\ell+1$ modified Okamoto-Schnorr signatures will, since they are tested only with the other two verification equations.

There's still one problem: if $\mathcal{A}_{\mathrm{P}_{OSC}}$ performs the "public verification" of the random choice of the generators [25], $\mathcal{S}$ might not be able to generate the correct data for this verification [26]. However, we can assume that the verification is not essential for the attack: the attack should work against an honest signer, and generators chosen by such a signer would pass the test; thus, $\mathcal{A}_{\mathrm{P}_{OSC}}$ wouldn't gain any new knowledge, since we can require that the verification doesn't give him other information than that the generators were indeed chosen correctly. Therefore, we may assume that $\mathcal{A}_{\mathrm{P}_{OSC}}$ doesn't perform the verification; otherwise, we know that another attacker exists who is identical to $\mathcal{A}_{\mathrm{P}_{OSC}}$, except that he doesn't perform the verification.

If in doubt about all this, we can simply let a trusted third party (e.g. the governmental institution that supervises banks) perform the verification, thus not allowing $\mathcal{A}$ to perform it. We believe that this question is not of great importance for the security of our scheme against coin forgery.

For all other aspects, $\mathcal{S}$ is as indistinguishable to $\mathcal{A}_{\mathrm{P}_{OSC}}$ as the simulator constructed under our "magic assumption" in Subsection 6.4.1 was to $\mathcal{A}_{\mathrm{P}_{OS}}$. For the "checker function", $\mathcal{S}$ simply passes the corresponding values $(I, \beta_I, \gamma_I, \delta_I, \mu_I, \nu_I, \alpha_{h,I})$ on to Σ. Then, if $\mathcal{A}_{\mathrm{P}_{OSC}}$ exists, with nonnegligible probability,

[24] Otherwise, $\mathcal{A}_{\mathrm{P}_{OSC}}$ could solve the Decision Diffie-Hellman problem: e.g., given the generator g and $g_1 = g^{b_1}; h = g^{\kappa}$, distinguishing whether $g_3 = h^{b_1} = g^{\kappa b_1}$ is precisely that problem.

[25] See "System Setup".

[26] This depends on how the generators in the real system are created and on how the verification is performed. If the verification makes use of some one-way function, $\mathcal{S}$ won't be able to trick $\mathcal{A}_{\mathrm{P}_{OSC}}$ into believing that the generators were chosen randomly.

the probabilistic polynomial-time Turing machine $\mathcal{A}$ from Figure 6.13 will succeed in an $(\ell, \ell+1)$-forgery in the modified Okamoto-Schnorr signature scheme with checker. As this was proven to be infeasible, we conclude that our scheme is secure against this forgery in the random oracle model.

Double-spending is no issue in an on-line system.

Alternative Approach for Proving Security Against $(\ell, \ell+1)$-Forgery

After the problems we had trying to prove the security of protocol P_{OS} by use of black box reduction, similar problems for a proof for the whole e-cash scheme seemed possible. Therefore a different approach to prove the security of the scheme against $(\ell, \ell+1)$-forgery was thought of. The question was whether the forking lemma for blind signatures (Lemma 6.7) could be used to prove the security of our scheme.

The authors of [PS96b] stated that the forking lemma can be applied to any blind signature protocol resulting from a witness indistinguishable identification protocol. The Okamoto-Schnorr signature scheme has this property and is, as we put it, "secret key indistinguishable". Our e-cash system, however, is not. This is due to the fact that $\mathcal{U}$ gets not only $\mathcal{B}$'s public key $y := g^{-r}h^{-s}$, but also the value $z_w := h_{w_1}^{-r}h_{w_2}^{-s}$. Hence, the use of different secret keys (r, s), (r', s') can be detected; actually, the values y, z_w are already sufficient to compute the secret key that was used to create them, under the assumption that one can compute discrete logarithms.

So there is no unconditional indistinguishability; what we can hope for is that

1. our scheme is *computationally* "secret key indistinguishable",
2. this suffices to be able to apply the forking lemma for blind signatures.

What does the indistinguishability precisely mean? Let $\mathcal{B}$ know two secret keys $(r, s) \neq (r', s')$ such that $y = g^{-r}h^{-s} = g^{-r'}h^{-s'}$, and let $\mathcal{U}$ and $\mathcal{B}$ execute the withdrawal protocol twice. $\mathcal{U}$ sends h_{w_1}, h_{w_2} and obtains the value z_w during the first execution; let h'_{w_1}, h'_{w_2}, and z'_w be the corresponding values from the second protocol execution. Now, our scheme is "secret key indistinguishable" if it is infeasible for $\mathcal{U}$ to know whether $\mathcal{B}$ computed both z_w and z'_w using the same secret key.

We believe this is the case; regrettably we couldn't find any problem, e.g. the Decision Diffie-Hellman problem, that can be reduced to our problem. Obviously, our assumption can only hold if $\mathcal{U}$ isn't allowed to choose the values h_{w_1}, h_{w_2}, h'_{w_1}, h'_{w_2} in some specific manner. Choosing $h_{w_1} = h'_{w_1}$ and $h_{w_2} = h'_{w_2}$ will clearly instantly reveal whether the same secret key is used to compute z_w and z'_w.

Under the assumption that it is infeasible for an honest user $\mathcal{U}$ to distinguish whether $\mathcal{B}$ used the same secret key, we're interested in whether the forking lemma for blind signatures can be applied. We believe that the proof of the lemma can be modified to fit our scheme at least up to Lemma 6.8; there, of course, we get a real problem. The two views of the attacker won't be identical. Due to lack of time (and because the black box reduction was so much easier) we're not investigating this further here.

Security for $\mathcal{S}$

As the system is on-line, coins paid to $\mathcal{S}$ should be hard to steal or extort (deposit takes place immediately after payment). There are no means to prevent false accusations of double-depositing. However, this is not a serious problem ($\mathcal{B}$ wouldn't usually make such false accusations, as there is no money to be gained) and can be solved easily (by including a signature by $\mathcal{S}$ in every deposit request).

Security for $\mathcal{T}$

It should be impossible to obtain untraceable coins. A coin becomes untraceable if $\mathcal{U}$ uses, instead of ϕ, another blinding factor ϕ' for the computation of h_{p_1}, h_{p_2}, and z_p. The values h_{p_1}, h_{p_2} contained in the resulting coin would then have to be of the form [27]

$$h_{p_1} = g_1 g_2^{\phi'} \quad \text{and} \quad h_{p_2} = g_3 g_4^{\phi'}$$

[27] Recall that the proof V verified during payment ensures this.

such that $\phi' \neq \phi$; thus, anonymity revocation with the value $d := y_{\mathcal{T}}^{\phi}$ would yield $g_1 g_2^{\phi} \neq h_{p_1}$. In protocol P_{OSC}, $\mathcal{U}$ obtains the values $a_{g,i}, a_{h,i}$; to obtain a valid but untraceable coin, he has to compute a value $\alpha_{h,i}$ such that at the end of the protocol, the verification equation

$$\alpha_{h,i} \stackrel{?}{=} h_{p_1}^{\rho} h_{p_2}^{\sigma} z_p^{\varepsilon}$$

holds, i.e.

$$\alpha_{h,i} \stackrel{?}{=} h_{p_1}^{t_i+e_i r} h_{p_2}^{u_i+e_i s} z_p^{\varepsilon_i}.$$

This will be the case if $\alpha_{h,i}$ is computed as

$$\alpha_{h,i} := h_{p_1}^{t_i} h_{p_2}^{u_i} h_{p_1}^{\beta_i} h_{p_2}^{\gamma_i} z_p^{\delta_i},$$

thus replacing $a_{h,i}^{\phi} = h_{w_1}^{t_i \phi} h_{w_2}^{u_i \phi}$ by $h_{p_1}^{t_i} h_{p_2}^{u_i} = h_{w_1}^{t_i \phi'} h_{w_2}^{u_i \phi'}$. This seems the easiest way to obtain a valid but untraceable coin.

We had a similar situation in the on-line system from [CMS96]; as we stated on Page 41, it was proven in [Cam98] that an algorithm capable of obtaining an untraceable coin in that system would solve the Diffie-Hellman problem. We tried to obtain an equivalent argument for our scheme, i.e. to show that if there is an algorithm which on input $g, h, g_1, \ldots, g_4, \phi, \phi', a_{g,i}, a_{h,i}$ outputs

$$h_{p_1}^{t_i} h_{p_2}^{u_i} = (g_1 g_2^{\phi'})^{t_i} (g_3 g_4^{\phi'})^{u_i},$$

then we can use this algorithm to solve some hard problem. It seems that there's an even easier way to see why such an algorithm can't exist:

Recall that $a_{h,i} := h_{w_1}^{t_i} h_{w_2}^{u_i}$; clearly, we have "witness indistinguishability" for $a_{h,i}$, i.e. there's no way to distinguish the q different pairs $(t_i', u_i') \in (\mathbf{Z}_q \times \mathbf{Z}_q)$ that all result in this particular value $a_{h,i}$. For an honest $\mathcal{U}$, it's of no relevance which values t_i, u_i $\mathcal{B}$ actually uses, since $\mathcal{U}$ performs calculations only with the product $a_{h,i}$. A dishonest $\mathcal{U}$ who tries to obtain an untraceable coin, however, would have to compute the value $h_{p_1}^{t_i} h_{p_2}^{u_i}$ that replaces $a_{h,i}^{\phi}$ with exactly the same values t_i, u_i that were used for computing $a_{h,i}$. Suppose he could compute the set

$$H := \{h_{p_1}^{t_i'} h_{p_2}^{u_i'} \in \mathbf{Z}_q | a_{h,i} := h_{w_1}^{t_i'} h_{w_2}^{u_i'}\};$$

then the probability that he would guess $h_{p_1}^{t_i} h_{p_2}^{u_i}$, i.e. the unique element out of this set that results in a valid signature, is $1/q$, and hence negligible.

But we have to consider that $\mathcal{U}$ also has the information $a_{g,i} := g^{t_i} h^{u_i}$, and thus there is a unique pair (t_i, u_i) that represents both $a_{g,i}$ and $a_{h,i}$. Therefore we'll make a reduction: if an algorithm was capable of computing the correct value $h_{p_1}^{t_i} h_{p_2}^{u_i}$ with the additional information $a_{g,i}$, we could use this algorithm to obtain untraceable coins in the system from [CMS96] like this:

In that system, we get the values

$$\tilde{t}_g := g^{\tilde{r}}$$

$$\tilde{t}_h := h_w^{\tilde{r}} = g_1^{1/\alpha} g_2,$$

and need to compute the value $h_p^{\tilde{r}} = (g_1 g_2^{\alpha'})^{\tilde{r}}$. Clearly, we also have the values α, α'. We set the inputs for our algorithm as follows:

$$\begin{array}{rcl}
(g, g_1, g_2) & := & (g, g_1, g_2) \text{ as in the system from [CMS96]} \\
h = g_3 = g_4 & := & 1 \\
\phi & := & \alpha \\
\phi' & := & \alpha' \\
a_{g,i} & := & \tilde{t}_g = g^t h^u \text{ for some } t, u \in \mathbf{Z}_q \\
a_{h,i} & := & \tilde{t}_h = g_1^{1/\phi} g_2 g_3^{1/\phi} g_4
\end{array}$$

Now, with these inputs our algorithm computes

$$h_{p_1}^{t} h_{p_2}^{u} = (g_1 g_2^{\phi'})^{t} (g_3 g_4^{\phi'})^{u} = (g_1 g_2^{\alpha'})^{t} 1^u = h_p^{\tilde{r}}.$$

The last equality holds because

$$g^{\tilde{r}} =: \tilde{t}_g = g^t h^u = g^t 1^u \quad \Rightarrow \quad t = \tilde{r}.$$

Thus, we can compute the value needed to obtain untraceable coins in the system from [CMS96], and this implies the ability to solve the Diffie-Hellman problem. As this is believed to be infeasible, we conclude that it is also infeasible to obtain untraceable coins in our scheme.

6.4.3 Self-Escrowing

Our system fulfills the requirements R1, R2, R3 from Subsection 5.1.1: $\mathcal{T}$ being passive, he doesn't participate in the regular protocols (withdrawal and payment), but performs only anonymity revocation; $\mathcal{T}$ is therefore trusted only for tracing (R1). During withdrawal, $\mathcal{T}$ is passive; $\mathcal{U}$ encrypts the tracing information d and proves its correct construction to $\mathcal{B}$ (R2). The public key of $\mathcal{T}$ isn't needed during payment or deposit (R3).

The modifications to the our scheme are analogous to those to the system from [CMS96]:

At account opening, $\mathcal{U}$ provides an additional public key $pk_{trace} := g_2^{sk_{trace}}$ and proves knowledge of the corresponding secret key sk_{trace}. $\mathcal{U}$ signs pk_{trace} with his regular signature key to prevent dispute about it.

At withdrawal, the encrypted tracing information $d := y_{\mathcal{T}}^{\phi}$ is replaced by $d := pk_{trace}^{\phi}$. Consequently, the proofs U_1, U_2 about the correct construction of h_{w_1}, h_{w_2}, and d become

$$U_1 := Proof_{EQLOG}(\epsilon, g_1, (h_{w_1}/g_2), d, pk_{trace}),$$

$$U_2 := Proof_{EQLOG}(\epsilon, g_3, (h_{w_2}/g_4), d, pk_{trace}).$$

Since $log_{g_1}(h_{w_1}/g_2) = log_d(pk_{trace})$ and $log_{g_3}(h_{w_2}/g_4) = log_d(pk_{trace})$, $\mathcal{B}$'s verification of U_1 and U_2 will succeed.

At payment, pk_{trace} is never handed to $\mathcal{S}$. Neither is any other value that would make it possible for $\mathcal{S}$ or $\mathcal{B}$ or both to link the payment to the withdrawal of the coin. Thus, the self-escrow-based scheme preserves $\mathcal{U}$'s anonymity just as well as the escrow-based one.

Tracing is done as in our escrow-based scheme: $\mathcal{B}$ retrieves the encrypted tracing information $d := pk_{trace}^{\phi} = (g_2^{sk_{trace}})^{\phi}$. $\mathcal{U}$ then computes $g_1 d^{1/sk_{trace}} = g_1 g_2^{\phi} = h_{p_1}$.

6.5 Off-line Payments

We'all adapt the modifications that transformed the on-line system from [CMS96] to an off-line scheme to our system; the withdrawal and payment protocols are slightly modified, allowing $\mathcal{B}$ to identify double-spenders without the help of $\mathcal{T}$. The anonymity of honest users is maintained.

The proof V is redefined; we assume that it was implemented as in Subsection 2.7.4. Then, to understand how the identification of double-spenders works, one has to keep in mind that V is an extended Schnorr signature. During its generation, the signer $\mathcal{U}$ chooses a random value r to compute c as $c := \mathcal{H}(m||g_1||g_2||h_1||h_2||g_1^r||g_2^r)$. If two different messages are signed using the same value r, the secret key can be computed from the two signatures. Therefore it suffices to force $\mathcal{U}$ to compute V using the same r for a given coin. Double-spending will then result in $\mathcal{B}$ being able to compute the secret key used to create V, which is ϕ; then $\mathcal{B}$ can look up $d = y_{\mathcal{T}}^{\phi}$ in the withdrawal database, and thus identify $\mathcal{U}$.

$\mathcal{U}$ has to choose r_p (which plays the role of r) and use $t_p := g_2^{r_p}$ instead of the coin number $c\#$ as the message m in protocol P$_{OSC}$ during withdrawal, i.e. $\mathcal{U}$'s input to protocol P$_{OSC}$ changes from $(y, c\#, h_{w_1}, h_{w_2}, \phi)$ to $(y, t_p, h_{w_1}, h_{w_2}, \phi)$. Since this is the only modification to the withdrawal protocol, the whole new protocol is not shown here.

The new payment protocol, however, is presented in Figure 6.15; $\mathcal{U}$ sends $(h_{p_1}, h_{p_2}, z_p, \alpha_g, \alpha_h, \varepsilon, \rho, \sigma, t_p)$ to $\mathcal{S}$. After verifying the modified Okamoto-Schnorr signature $sig := (\alpha_g, \alpha_h, \varepsilon, \rho, \sigma)$ resulting from protocol

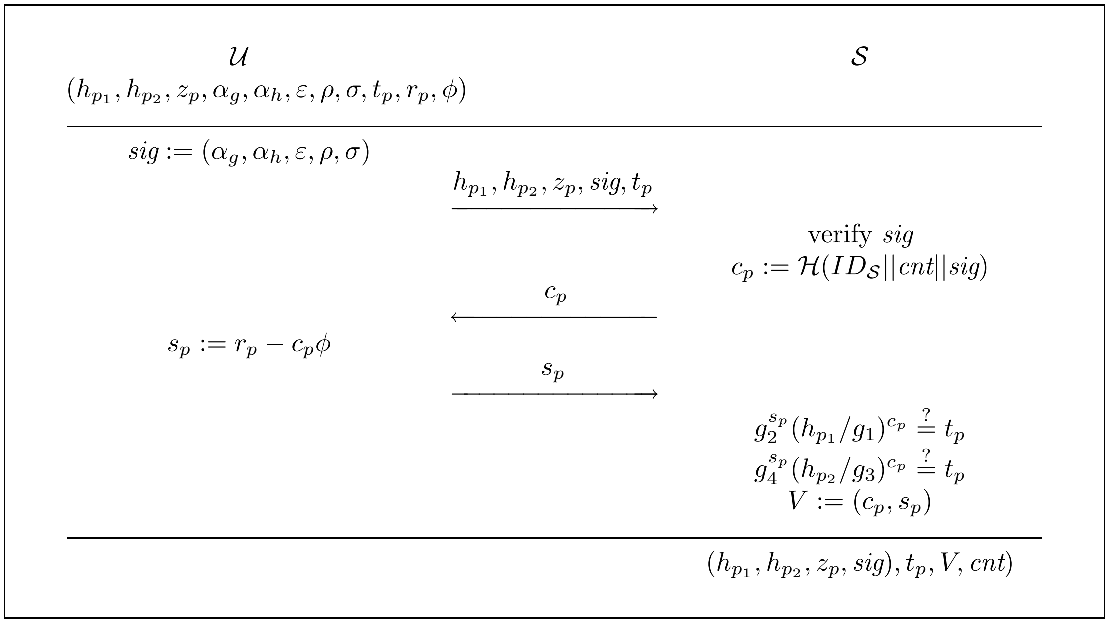

$\mathcal{U}$
$\mathcal{S}$
$(h_{p_1}, h_{p_2}, z_p, \alpha_g, \alpha_h, \varepsilon, \rho, \sigma, t_p, r_p, \phi)$
$sig := (\alpha_g, \alpha_h, \varepsilon, \rho, \sigma)$
$h_{p_1}, h_{p_2}, z_p, sig, t_p$
verify sig
$c_p := \mathcal{H}(ID_{\mathcal{S}}||cnt||sig)$
c_p
$s_p := r_p - c_p\phi$
s_p
$g_2^{s_p}(h_{p_1}/g_1)^{c_p} \overset{?}{=} t_p$
$g_4^{s_p}(h_{p_2}/g_3)^{c_p} \overset{?}{=} t_p$
$V := (c_p, s_p)$
$(h_{p_1}, h_{p_2}, z_p, sig), t_p, V, cnt)$

Figure 6.15: Off-line payment.

P$_{OSC}$, $\mathcal{S}$ provides $\mathcal{U}$ with the challenge $c_p := \mathcal{H}(ID_{\mathcal{S}}||cnt||sig)$ and obtains the response $s_p := r_p - c_p\phi$ (mod q). The verifications subsequently performed by $\mathcal{S}$ succeed if s_p was computed correctly, because then

$$g_2^{s_p}(h_{p_1}/g_1)^{c_p} = g_2^{r_p - c_p\phi}(h_{w_1}^{\phi}/g_1)^{c_p} = g_2^{r_p - c_p\phi} g_2^{c_p\phi} = g_2^{r_p} = t_p,$$

$$g_4^{s_p}(h_{p_2}/g_3)^{c_p} = g_4^{r_p - c_p\phi}(h_{w_2}^{\phi}/g_1)^{c_p} = g_2^{r_p - c_p\phi} g_4^{c_p\phi} = g_2^{r_p} = t_p.$$

The equalities $h_{p_i} = h_{w_i}^{\phi} = g_1 g_2^{\phi}$ hold for $i \in \{0, 1\}$ because *sig* proves that the withdrawal protocol was executed correctly.

The "new" proof V obtained by $\mathcal{S}$ is $V := (c_p, s_p)$; $\mathcal{S}$' final result $(h_{p_1}, h_{p_2}, z_p, sig), t_p, V, cnt)$ can be passed to $\mathcal{B}$ for deposit who will be able to verify both *sig* and V. The counter value *cnt* serves to protect $\mathcal{U}$ against $\mathcal{S}$ framing him as a double-spender; if *cnt* wasn't included, $\mathcal{S}$ could double-deposit the coin and say $\mathcal{U}$ paid twice with it. With *cnt*, $\mathcal{B}$ will only believe this if the values *cnt*, *cnt'* in the two payment transcripts differ, i.e. if $\mathcal{U}$ actually responded to two different challenges.

As for the off-line version of the scheme from [CMS96], including *sig* links V to a specific withdrawal and therefore to this specific coin.

Double-spender Identification

If $\mathcal{U}$ spends a coin twice, $\mathcal{B}$ will end up with two different challenge-response pairs (c_p, s_p), (c'_p, s'_p) related to the coin. $\mathcal{B}$ then computes

$$\begin{aligned} s_p - s'_p &= r_p - c_p\phi - r_p + c'_p\phi = \phi(c'_p - c_p) \\ \Rightarrow \quad \phi &= \frac{s_p - s'_p}{c'_p - c_p} \\ \text{and} \quad d &= y_{\mathcal{T}}^{\phi}, \end{aligned}$$

and looks d up in the withdrawal database, thus identifying $\mathcal{U}$.

Changes in Security

Avoiding double-spender identification requires the ability to forge the signature V; this is believed to be infeasible.

$\mathcal{U}$ can be falsely accused of overspending by $\mathcal{B}$ if $\mathcal{B}$ can present two different signatures V for one coin although $\mathcal{U}$ spent the coin only once. This means being able to forge V, which is believed to be infeasible. Should $\mathcal{U}$ indeed overspend, then $\mathcal{B}$ could compute ϕ and collude with some $\mathcal{S}$ to further overspend the coin.

Theft or extortion of coins paid to $\mathcal{S}$ is prevented by including $Id_{\mathcal{S}}$ in the payment transcript and thus making the coin depositable only by $\mathcal{S}$.

For all other aspects, the off-line scheme is as secure as the on-line system.

6.5.1 Self-Escrowing

Implementing self-escrowing in the off-line scheme works just as for the on-line system.

Chapter 7

Conclusion

In the previous chapters, we presented various escrow-based e-cash systems and investigated their security. Self-escrowing against user blackmailing was generalized and we showed that for schemes with a passive trutee, it can be implemented by replacing the trustee by the user. We then addressed ways to provide formal proofs of security against coin forgery while keeping the schemes efficient. Based on a provably secure signature scheme, we built an on-line escrow-based scheme which can be made self-escrow-based. Moreover, both variants of it can be modified so as to provide off-line payments. The security of our scheme against coin forgery was proven in the random oracle model by black box reduction.

7.1 Open Problems

As far as the security of our scheme against other attacks than coin forgery is concerned, it would be interesting to prove it in a rigorous way, i.e. without "plausibility arguments", but with formal proofs instead.

Furthermore, the security of our scheme against coin forgery should be provable by application of an adapted version of the forking lemma, instead of the reduction we performed.

Finally, it remains open whether the efficiency of our scheme can be improved without loss of security.

7.2 Other Approaches

In the following, we'll briefly present approaches to anonymity control which are different from those investigated in this thesis.

7.2.1 Flow Control [STS99a]

In [STS99a], it was claimed that anonymity control is mainly desirable to prevent the (illegal) flow of large amounts of money, while untraceability and unlinkability of small cash amounts can be tolerated as with paper-based cash. The suggested "flow control" approach therefore allows each user to spend only a limited amount of e-cash anonymously in a given time frame. To prevent the accumulation of many small amounts of e-cash to a large amount, the electronic coins have to be non-transferable. The basic idea for achieving this is to make the coins spendable only by someone who knows a so-called nontransferability secret. This secret has to be of such importance that no user would give it away (e.g. a digital signature key that can be used to sign documents of legal relevance); alternatively, the secret can be hidden altogether from the user (e.g. stored in a tamper-resistant device which in turn could require biometric authentication). As the authors of [STS99a] pointed out, no system can be perfectly non-transferable, but as the amount of work a criminal would have to do to accumulate a reasonable amount of anonymous e-cash rises, such a scenario gets more and more improbable.

Both an on-line and an off-line scheme were presented; the latter is based on [Bra93].

7.2.2 Auditable Anonymous E-Cash [STS99b]

In [STS99b], a fully anonymous electronic cash system was presented which isn't signature-based. Moreover, it is auditable, i.e. banks can't secretly issue coins; this implies that blackmailing a bank won't work, because even if the bank remains silent, the auditing authority will notice the attack. This also prevents insider attacks or theft of secret keys of a bank (actually, there are no secret keys of banks required for issuing coins). It is of course assumed that coins can be invalidated.

The bank maintains a public database based on a list of valid coins, and the user proves during payment that he knows one of these coins, without showing it. Thus, his anonymity is preserved. The public database consists of a forest of so-called hash trees, which are binary trees such that for a given hash function $\mathcal{H}$ the following holds: for any vertex v with two children v_1, v_2, we have $v = \mathcal{H}(v_1, v_2)$. The coins consist of a value z of a leaf of a tree together with values that form a so-called hash chain from z to the root of the tree.

The roots of all trees in the forest are broadcast at regular intervals to all shops. If a shop chooses to receive these updates less frequently, the only limitation is that it won't be able to accept coins withdrawn recently.

The system from [STS99b] was stated to be "theoretically efficient but not yet practical."

Bibliography

[AJSW97] N. Asokan, P. A. Janson, M. Steiner, and M. Waidner. The state of the art in electronic payment systems. *Computer*, 30(9):pages 28–35, September 1997.

[Bon98] D. Boneh. The Decision Diffie-Hellman problem. In *Third Algorithmic Number Theory Symposium*, number 1423 in Lecture Notes in Computer Science, pages 48–63. Springer-Verlag, Berlin Germany, 1998.

[BP89] H. Bürk and A. Pfitzmann. Digital payment systems enabling security and unobservability. *Computers & Security*, 8(5):pages 399–416, August 1989.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *1st ACM Conference on Computer and Communications Security*. ACM Press, Fairfax, Virginia, November 1993. Also appeared (in identical form) as IBM RC 19619 (87000) 6/22/94.

[Bra93] S. Brands. Untraceable off-line cash in wallets with observers. In D. R. Stinson, editor, *Advances in Cryptology – CRYPTO ' 93*, volume 773 of *Lecture Notes in Computer Science*, pages 302–318. Springer-Verlag, Berlin Germany, 1993.

[Cam98] J. L. Camenisch. Group signature schemes and payment systems based on the discrete logarithm problem. Ph.D. dissertation, ETH Zürich, 1998.

[CEvdG88] D. Chaum, J.-H. Evertse, and J. van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In D. C. W. L. Price, editor, *Conference on the Theory and Application of Cryptographic Techniques*, volume 304 of *LNCS*, pages 127–142. Springer, Berlin, April 1988. ISBN 3-540-19102-X.

[CFN88] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In S. Goldwasser, editor, *Proc. CRYPTO 88*, pages 319–327. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 403.

[Cha83] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, *Advances in Cryptology: Proceedings of Crypto 82*, pages 199–203. Plenum Press, 23–25 August 1983.

[CMS96] J. L. Camenisch, U. Maurer, and M. A. Stadler. Digital payment systems with passive anonymity-revoking trustees. In *ESORICS: European Symposium on Research in Computer Security*. LNCS, Springer-Verlag, 1996.

[CP92] D. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *Proc. CRYPTO 92*, pages 89–105. Springer-Verlag, 1992. Lecture Notes in Computer Science No. 740.

[CPS95] J. L. Camenisch, J.-M. Piveteau, and M. A. Stadler. Fair blind signatures. In L. Guilloy and J.-J. Quisquater, editors, *Advances in Cryptology – EUROCRYPT ' 95*, Lecture Notes in Computer Science, pages 209–219. Springer-Verlag, Berlin Germany, 1995.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):pages 644–654, November 1976.

[dST98] A. de Solages and J. Traoré. An efficient fair off-line electronic cash system with extensions to checks and wallets with observers. *Lecture Notes in Computer Science*, 1465:pages 275–295, 1998.

[ElG85] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):pages 473–481, 1985.

[Fer93] N. T. Ferguson. Single term off-line coins. Technical Report CS-R9318, CWI - Centrum voor Wiskunde en Informatica, December 31, 1993.

[FO97] E. Fujisaki and T. Okamoto. Practical escrow cash systems. *Lecture Notes in Computer Science*, 1189:pages 33–48, 1997.

[FO98] E. Fujisaki and T. Okamoto. Practical escrow cash schemes. *IEICE Trans. Fundamentals*, E-81A:pages 11–19, 1998.

[FS86] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology—CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987, 11–15 August 1986.

[FS90] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *22nd Annual Symposium on Theory of Computing (STOC)*, pages 416–426. ACM Press, Baltimore, MD, USA, May 1990.

[FTY96] Y. Frankel, Y. Tsiounis, and M. Yung. "Indirect discourse proofs": Achieving efficient fair off-line E-cash. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology—ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 286–300. Springer-Verlag, Kyongju, Korea, 3–7 November 1996.

[FTY98] Y. Frankel, Y. Tsiounis, and M. Yung. Fair off-line e-cash made easy. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*. LNCS, Springer-Verlag, 1998.

[GM92] D. M. Gordon and K. S. McCurley. Massively parallel computation of discrete logarithms. In E. F. Brickell, editor, *Advances in Cryptology—CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 312–323. Springer-Verlag, 1993, 16–20 August 1992.

[McC90] K. S. McCurley. The discrete logarithm problem. In C. Pomerance, editor, *Cryptology and Computational Number Theory*, volume 42 of *Proceedings of Symposia in applied Mathematics*. American Mathematical Society, 1990.

[MvOV97] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. Boca Raton, 1997.

[Oka93] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *Advances in Cryptology – CRYPTO ' 92*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer-Verlag, 1993.

[Pfi98] B. Pfitzmann. Higher cryptographic protocols. Course material, Universität des Saarlandes, Germany, http://www-krypt.cs.uni-sb.de, 1998.

[Pfi99] B. Pfitzmann. Security (Sicherheit) - additional course material. Course material, Universität des Saarlandes, Germany, http://www-krypt.cs.uni-sb.de, 1999.

[Poi98] D. Pointcheval. Strengthened security for blind signatures. In K. Nyberg, editor, *Advances in Cryptology: EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*. Springer, 1998.

[PS96a] D. Pointcheval and J. Stern. Security proofs for signature schemes. In U. Maurer, editor, *Advances in Cryptology—EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer-Verlag, 12–16 May 1996.

[PS96b] D. Pointcheval and J. Stern. Provably secure blind signature schemes. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology—ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 252–265. Springer-Verlag, Kyongju, Korea, 3–7 November 1996.

[PS97] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. Manuscript, 1997.

[PS00] B. Pfitzmann and A.-R. Sadeghi. Self-escrowed cash against user blackmailing. Technical report, Universität des Saarlandes, Germany, 2000.

[Rom90] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In B. Awerbuch, editor, *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, pages 387–394. ACM Press, Baltimore, MY, May 1990. ISBN 0-89791-361-2.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosytems. *Commun. ACM*, 21(2):pages 120–126, 1978.

[Sch91] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):pages 161–174, 1991.

[STS99a] T. Sander and A. Ta-Shma. Flow control: A new approach for anonymity control in electronic cash systems. In *Financial cryptography: Third International Conference, FC '99, Anguilla, British West Indies, February 22–25, 1999: proceedings*, volume 1648 of *Lecture Notes in Computer Science*, pages 46–61. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1999. ISBN 3-540-66362-2 (softcover).

[STS99b] T. Sander and A. Ta-Shma. Auditable, anonymous electronic cash (extended abstract). In *Proceedings of Crypto*, 1999.

[TY98] Y. Tsiounis and M. Yung. On the security of ElGamal-based encryption. *Lecture Notes in Computer Science*, 1431:pages 117–??, 1998.

[vSN92] S. von Solms and D. Naccache. On blind signatures and perfect crimes. *Computers and Security*, 11(6):pages 581–583, October 1992.

[Zei96] E. Zeidler, editor. *Teubner-Taschenbuch der Mathematik*. B.G. Teubner Stuttgart; Leipzig, 1996.

Index