Technical Report No. 260, March 1997, Dept. of Computer Science, ETH Zurich.
Electronically available from: ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/

Proof Systems for General Statements about Discrete Logarithms

Jan Camenisch

Dept. of Computer Science
Haldeneggsteig 4
ETH Zurich
CH-8092 Zurich, Switzerland
e-mail: camenisch@inf.ethz.ch

Markus Stadler

Union Bank of Switzerland
Ubilab
Bahnhofstrasse 45
CH-8021 Zurich, Switzerland
e-mail: Markus.Stadler@ubs.com

Abstract

Proof systems for knowledge of discrete logarithms are an important primitive in cryptography. We identify the basic underlying techniques, generalize these techniques to prove linear relations among discrete logarithms, and propose a notation for describing complex and general statements about knowledge of discrete logarithms. This notation leads directly to a method for constructing efficient proof systems of knowledge.

1 Introduction

Many complex cryptographic systems, such as payment systems (e.g. see [1, 2, 4]) and voting schemes [11], are based on the difficulty of the discrete logarithm problem. These systems make use of various minimum-disclosure proofs of statements about discrete logarithms [13, 7, 6, 10]. Typical examples are efficient proofs of knowledge of a discrete logarithm which are based on Schnorr's digital signature scheme [18] and systems for proving the equality of two discrete logarithms, as used in [8].

The goal of this paper is to identify the basic techniques for proving statements about discrete logarithms, to generalize them, and to define a formal notation for specifying statements about discrete logarithms that can be proved using these generalized techniques. In particular, the notation allows to define statements about the knowledge of discrete logarithms, about (linear) relations among them, and about monotone boolean functions whose atoms are also statements. This notation then leads to a method for deriving efficient proof systems from specifications.

Similar methods for constructing complex proof systems have already been presented by De Santis et al. [12] and independently by Cramer et al. [10]. In particular, given proof systems for single statements, they show how to construct a proof system for any monotone boolean formula over these statements. Although we restrict ourselves to statements about discrete logarithms (and representations), our method is a generalization of [12, 10], because it also includes the possibility to prove relations among witnesses (e.g. discrete logarithms). For instance, using the methods of [12, 10], a proof of equality of two discrete logarithms cannot be derived from simple proofs of knowledge of discrete logarithms.

In Section 2 we briefly describe the discrete logarithm and the representation problem and define some notations. In Section 3 we define proofs of knowledge informally and present some examples of discrete logarithm-based proofs. In Section 4 the notation for specifying statements about discrete logarithms is defined and explained. Then we show in Section 5 how, given such a specification, an efficient proof system can be constructed (an example can be found in Section 6). The paper is concluded in Section 7 with a discussion about possible improvements and open problems.

2 Preliminaries

Let G be a finite cyclic group of prime order q and let $g, g_1, \ldots, g_k \in G$ be generators of G, for some $k > 0$ (note that the primality of q is not a necessary condition but is here assumed for simplicity). The *discrete logarithm* of an element $y \in G$ to the base g is the unique integer x, $0 \leq x < q-1$, for which $y = g^x$. The discrete logarithm is also called the *index* of y with respect to the base g. An *index tuple* of y with respect to the bases $g_1, \ldots, g_k$ is a k-tuple $(x_1, \ldots, x_k)$, with $0 \leq x_i < q-1$ for $i = 1, \ldots, k$ and $\prod_{i=1}^{k} g_i^{x_i} = y$. The index tuple $(x_1, \ldots, x_k)$ is also called a *representation* of y with respect to $g_1, \ldots, g_k$. See [1] for further discussions about the representation problem.

Let us now define some notation that will be used throughout the paper. The concatenation of the strings α and β is denoted by $\alpha \| \beta$. The expression $\xi \in_R X$ means that ξ is chosen randomly from the (finite) set X according to the uniform distribution. Finally, let $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\ell$, denote a collision resistant hash function that maps the binary representation of the argument to a binary string of length ℓ (e.g. $\ell = 128$).

3 Proofs of Knowledge

Informally, a proof of knowledge allows a *prover* to convince (prove to) a *verifier* that he knows a solution of a hard-to-solve problem, such that the following properties hold:

- an honest prover, knowing a solution, can successfully convince the verifier (completeness),
- with overwhelming probability, a cheating prover, not knowing any solution, will fail to convince the verifier (soundness), and
- the verifier obtains no useful information about the solution the prover knows (there are different definitions about what "obtaining no useful information" means, for instance zero-knowledge, witness-hiding, and minimum-disclosure).

Proofs of knowledge have been introduced and defined formally in [13], but we will also call systems proofs of knowledge if they do not meet the strong requirements of [13]. It has been shown that proofs of knowledge exist for a large class of problems [14, 3]. However, efficient proofs have been found only for some number-theoretic problems such as RSA-inversion and computing discrete logarithms [15, 7, 6, 18].

Particularly, proofs of knowledge of discrete logarithms and of representations are important ingredients of many cryptographic systems, from simple identification and signature schemes up to complex electronic voting and digital payment systems. In the first example we will present a simple proof of knowledge of a discrete logarithm.

Example 1. To prove the knowledge of the discrete logarithm of $y = g^x$ to the base g, the prover computes the following values:

1. $v \in_R \mathbb{Z}_q$, $t = g^v$

2. $c = \mathcal{H}(g, y, t)$
3. $r = v - cx \pmod{q}$

The values t, c, and r are called *commitment*, *challenge*, and *response*, respectively. The resulting proof is the pair (c, r) and can be verified (by everyone) by first reconstructing the commitment $t' = g^r y^c$ and then checking the equation $c \stackrel{?}{=} \mathcal{H}(g, y, t')$. This proof of knowledge is basically a Schnorr signature [18] for the message (g, y).

Let us briefly discuss the properties of the proof system of Example 1. First, it can easily be seen that an honest prover will always succeed in constructing a valid proof since

$$t' \;=\; g^r y^c \;=\; g^{v-cx} y^c \;=\; g^v \;=\; t$$

and therefore $c = \mathcal{H}(g, y, t')$. Second, assume that a cheating prover who does not know x was able to compute such proofs. Since the hash function is hard to invert, we can assume that the value $t' = g^r y^c$ was fixed before c was computed. It also seems necessary that when fixing the value t' the prover was prepared to compute a proof for many other possible challenges (otherwise the probability of success would be to small). But this means that the cheating prover could also compute different representations of t' to the bases g and y which implies the knowledge of x, the discrete logarithm of y to the base g, and this contradicts the assumption that the cheating prover does not know x. Note that a very similar idea, the so-called knowledge extractor, is used in [13] for defining the soundness property of interactive proofs of knowledge. Finally, under the assumption that $\mathcal{H}$ is a truly random function, it is possible to show that extracting the discrete logarithm from such proofs is as hard as computing discrete logarithms. If the protocol is executed interactively, i.e. the challenge is chosen by the verifier from a "small set of possible challenges", it can be proved to be zero-knowledge (which means that the verifier could have simulated all the information obtained in the protocol).

Based on this proof of knowledge of a discrete logarithm, several other systems have been proposed. One is a proof of the equality of two discrete logarithms (as used in [9] for a signature scheme). More generally, one can prove that two discrete logarithms satisfy a linear equation.

Example 2. To prove that the discrete logarithms of $y_1 = g_1^{x_1}$ and $y_2 = g_2^{x_2}$ to the bases g_1 and g_2, respectively, satisfy the linear equation $a_1 x_1 + a_2 x_2 = b \pmod{q}$, the prover proceeds as follows:

1. $(v_1, v_2) \in_R \{(u_1, u_2) \in \mathbb{Z}_q^2 \mid a_1 u_1 + a_2 u_2 = 0 \pmod{q}\}$, $t_1 = g_1^{v_1}$ and $t_2 = g_2^{v_2}$
2. $c = \mathcal{H}(g_1, y_1, g_2, y_2, a_1, a_2, b, t_1, t_2)$
3. $r_1 = v_1 - cx_1 \pmod{q}$ and $r_2 = v_2 - cx_2 \pmod{q}$

The resulting proof is (c, r_1, r_2) and can be verified by first reconstructing the commitments

$$t_1' = g_1^{r_1} y_1^c \;\text{ and }\; t_2' = g_2^{r_2} y_2^c$$

and then checking the equations

$$c \stackrel{?}{=} \mathcal{H}(g_1, y_1, g_2, y_2, a_1, a_2, b, t_1', t_2') \;\text{ and }\; a_1 r_1 + a_2 r_2 \stackrel{?}{=} -cb \pmod{q}.$$

In other words, the prover convinces the verifier that he or she

- knows the discrete logarithms of y_1 and y_2 to the bases g_1 and g_2, respectively,

- and that these logarithms satisfy the linear equation.

At a first glance, this looks like a new type of proof system, namely for proving properties of knowledge. However, these types of proofs can easily be modeled using the concept of proofs of knowledge.

The next example illustrates how different proofs can be combined. Given two problems X and Y and corresponding systems for proving the knowledge of solutions, it is trivial to construct a system for proving the knowledge of solutions to both X and Y (the two proof systems are simply executed in parallel). Proving the knowledge of a solution of problem X *or* of problem Y is more difficult because a verifier must not learn which solution the prover knows. A very interesting method for solving this problem was first proposed by Cramer et al. [10] and independently by De Santis et al. [12]. Let us demonstrate this method in Example 3.

Example 3. To prove the knowledge of the discrete logarithm of $y_1 = g^{x_1}$ to the base g_1 or the discrete logarithm of $y_2 = g^{x_2}$ to the base g_2, the prover proceeds as follows (assume that the prover knows x_2):

1. choose v_1, v_2, and $w \in_R \mathbb{Z}_q$ and compute $t_1 = y_1^w g_1^{v_1}$ and $t_2 = g_2^{v_2}$
2. $c = \mathcal{H}(g_1, y_1, g_2, y_2, t_1, t_2) \pmod q$
3. $c_1 = w$ and $c_2 = c - c_1 \pmod q$
4. $r_1 = v_1 \pmod q$ and $r_2 = v_2 - c_2 x \pmod q$

The resulting proof (c_1, c_2, r_1, r_2) can be verified by first reconstructing the commitments

$$t_1' = y_1^{c_1} g_1^{r_1} \quad \text{and} \quad t_2' = y_2^{c_2} g_2^{r_2}$$

and by checking the equation

$$c_1 + c_2 \stackrel{?}{=} \mathcal{H}(g_1, y_1, g_2, y_2, t_1', t_2') \pmod q.$$

The reason why this works is that the prover is "allowed to forge" one of the two proofs since he can choose the corresponding challenge before the commitment is computed; the other challenge is then determined by the hash function. The verifier, however, cannot decide which challenge was chosen and therefore obtains no information about which discrete logarithms the prover knows.

4 Knowledge Specification Sets

In the previous section we have presented the basic principles for proving knowledge about discrete logarithms and representations. These basic proofs can now be combined in order to prove the knowledge of solutions to more complex problems. We give a formal notation for specifying the knowledge that a party wants to prove. From this specification an efficient proof-system can be derived. Before describing this specification, we need to define the following notations.

Definition 4.1. Concatenation of tuples:
Let $a = (a_1,..., a_k)$ and $b = (b_1,..., b_\ell)$ be k- and ℓ-tuples, respectively. The concatenation of a and b, denoted $a \circ b$, is the tuple $(a_1,..., a_k, b_1,..., b_\ell)$.

Definition 4.2. Modified Cartesian Product:
Let A and B be sets of tuples. The modified Cartesian product of A and B, denoted $A \otimes B$, is the set of tuples

$$\{a \circ b \mid a \in A,\ b \in B\}.$$

Using this notation we define a set of values (which are the witnesses of the underlying NP language) and the proof of knowledge consists of proving the knowledge of at least one element of this set. We call such a set a *knowledge specification set* and its definition the *knowledge specification*. As mentioned in the introduction, we restrict ourselves to sets specifying knowledge about discrete logarithms and representations.

Definition 4.3. Knowledge specification set
Let G be a finite group of prime order q. Then a knowledge specification set for the group G is defined as follows:

- for any group elements g and y, the set

  $$\mathrm{DL}(g, y) := \{x \in \mathbb{Z}_q \mid y = g^x\}$$

  is a knowledge specification set.

- for any $k > 0$ and group elements $g_1, ..., g_k$, and y, the set

  $$\mathrm{REP}((g_1, ..., g_k), y) := \{(x_1, ..., x_k) \in \mathbb{Z}_q^k \mid y = \prod_{i=1}^{k} g_i^{x_i}\}$$

  is a knowledge specification set.

- for any $k > 0$ and values $a_1, ..., a_k,\ b \in \mathbb{Z}_q$, the set

  $$\mathrm{LE}((a_1, ..., a_k), b) := \{(x_1, ..., x_k) \in \mathbb{Z}_q^k \mid \sum_{i=1}^{k} a_i x_i = b \pmod{q}\}$$

  is a knowledge specification set.

- for knowledge specification sets A and B, the sets

  $$A \otimes B,\ A \cap B, \ \text{ and } \ A \cup B$$

  are also knowledge specification sets.

Remarks. The set $\mathrm{DL}(g, y)$ contains only one element and due to the discrete logarithm problem is it hard to compute this element for given y and g. The set $\mathrm{REP}((g_1, ..., g_k), y)$ can contain more than one element. However, it is hard to compute any other than the known representation if the bases are chosen in a random manner. Note that DL is just a special case of REP with $k = 1$. Furthermore, proving the knowledge of an element of a set $\mathrm{LE}((a_1, ..., a_l), b)$ is trivial because it is easy to compute a solution of the equation $\sum_{i=1}^{k} a_i x_i = b \pmod{q}$. Nevertheless, such set make sense when combined with other statements using the $\cap$-operator in order to express linear relations among several discrete logarithms or representations.

Let us now briefly discuss a few examples of knowledge specification sets. First, proving the knowledge of an element of the set $\mathrm{DL}(g, y)$ is equivalent to proving the knowledge of the discrete logarithm of y to the base g, and a set $\mathrm{REP}((g_1,\dots,g_k), y)$ corresponds with a proof of knowledge of a representation of y to the bases $g_1,\dots,g_k$.

Furthermore, the knowledge specification set of the proof in Example 2 can be defined as given below. It shows how linear equations can be used in knowledge specifications.

$$K = \underbrace{(\mathrm{DL}(g_1, y_1) \otimes \mathrm{DL}(g_2, y_2))}_{A} \cap \mathrm{LE}((a_1, a_2), b).$$

The set A contains exactly one pair consisting of the discrete logarithms of y_1 and y_2 to the bases g_1 and g_2, respectively. The intersection of A and $\mathrm{LE}((a_1, a_2), b)$ is non-empty if and only if the two logarithms satisfy the linear equation. Therefore, by proving the knowledge of an element of the set K, the prover indirectly proves that K is non-empty and thus the two discrete logarithms have the desired property.

Using also the union of sets, one can specify also more general statements, for instance for proving that two discrete logarithms are known and satisfy at least one of two linear equations:

$$A \cap (\mathrm{LE}((a_1, a_2), b) \;\cup\; \mathrm{LE}((d_1, d_2), e)).$$

Let us make a final remark about the intersection of knowledge specification sets. If one intersects sets containing tuples of different cardinalities, such as

$$\mathrm{DL}(g, y) \cap \mathrm{LE}((a_1, a_2), b),$$

it is obvious that the resulting set is empty and therefore no proof is possible. We will therefore assume in the sequel that such expressions are eliminated in knowledge specifications.

5 Construction of Proof Systems

In this section we show how to construct a proof-system for proving the knowledge of an element of an arbitrary knowledge specification set.

Transformation and Tree-Representation

Let F be a knowledge specification. By applying the transformations

$$\begin{array}{rl}
\text{I)} & (X \cup Y) \cap Z \to (X \cap Z) \cup (Y \cap Z) \\
\text{II)} & X \cap (Y \cup Z) \to (X \cap Y) \cup (X \cap Z) \\
\text{III)} & (X \cup Y) \otimes Z \to (X \otimes Z) \cup (Y \otimes Z) \\
\text{IV)} & X \otimes (Y \cup Z) \to (X \otimes Y) \cup (X \otimes Z)
\end{array}$$

to F and to its subexpressions, we can find a representation of F of the form

$$\tilde{F} = \bigcup_{i=1}^{m} \tilde{F}_i,$$

where the specifications $\tilde{F}_i$ contain no subexpressions of the form $X \cup Y$. From now on we will regard these $\tilde{F}_i$ as binary trees whose leaves are expressions of type REP, DL, or LE and whose inner nodes are of type $\cap$ or $\otimes$ (see Figure 1 for an example). The nodes are labeled as follows:

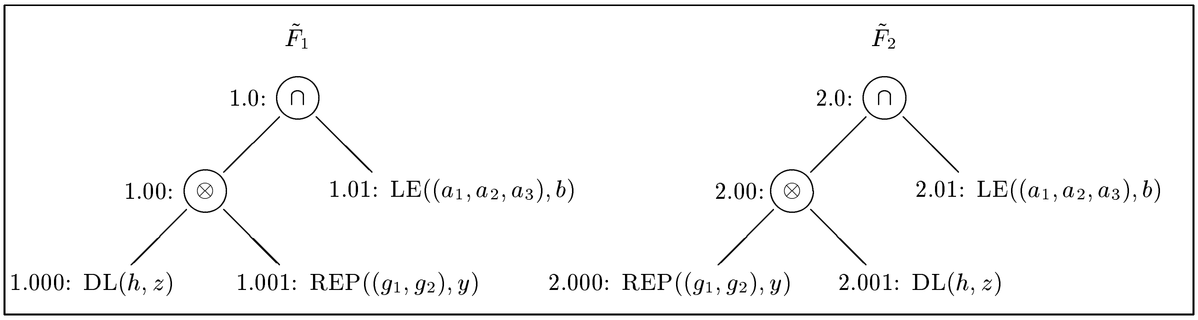

Figure 1: The set $F = \big((\mathrm{DL}(h,z) \otimes \mathrm{REP}((g_1,g_2),y)\big) \cup \big(\mathrm{REP}((g_1,g_2),y) \otimes \mathrm{DL}(h,z)\big)\big) \cap \mathrm{LE}((a_1,a_2,a_3),b)$ represented as a forest of two binary trees (see also Example 4 in the Appendix). The labels of the nodes are printed on the left side of each node.

- the root of tree $\tilde{F}_i$ is labeled $i.0$
- the left successor of a node labeled n is labeled $n\|0$
- the right successor of a node labeled n is labeled $n\|1$

Each node n in the tree $\tilde{F}_i$ is now assigned a pair (V_n, E_n), where V_n is a tuple of variables and E_n is a set of equations over the finite field $\mathbb{F}_q$ in the variables in V_n and in special variables w_i. These pairs are recursively defined as follows:

- if n is a leaf of type DL(g, y) then

$$V_n = (v_{n,1}) \text{ and } E_n = \emptyset$$

- if n is a leaf of type REP$((g_1,\dots,g_k), y)$ then

$$V_n = (v_{n,1},\dots,v_{n,k}) \text{ and } E_n = \emptyset$$

- if n is a leaf of type LE$((a_1,\dots,a_k), b)$ in the tree $\tilde{F}_i$ then

$$V_n = (v_{n,1},\dots,v_{n,k}) \text{ and } E_n = \{\sum_{j=1}^{k} a_j v_{n,j} = -w_i b\}$$

  Note that a single variable w_i is used for all nodes of type LE in the tree $\tilde{F}_i$.

- if n is an inner node of type $\otimes$ then

$$V_n = V_{n\|0} \circ V_{n\|1} \text{ and } E_n = E_{n\|0} \cup E_{n\|1}$$

- if n is an inner node of type $\cap$ then

$$V_n = V_{n\|0} \circ V_{n\|1} \text{ and } E_n = E_{n\|0} \cup E_{n\|1} \cup \bigcup_{j=1}^{k} \{V_{n\|0}(j) = V_{n\|1}(j)\}$$

In the last equation $V_{n\|\ldots}(j)$ denotes the j-th variable in the tuple $V_{n\|\ldots}$. Finally, let

$$V = V_{1.0} \circ \ldots \circ V_{m.0}, \quad W = (w_1, \ldots, w_m), \quad \text{and} \quad E = \bigcup_{i=1}^{m} E_{i.0}.$$

V and W are tuples of variables, and E is a system of linear equations (modulo q) in the variables in V and in W. In the sequel we will use the notation $E|_{W=(\ldots)}$, meaning that in E the variables of W are replaced by the corresponding values.

Constructing a proof for F

The (honest) prover knows an element $K \in F$ which must be contained in at least one of the sets $\tilde{F}_i$. Therefore there exists an index $\alpha \in \{1, \ldots, m\}$ such that $K \in \tilde{F}_\alpha$. Note that K is a tuple of elements of $\mathbb{Z}_q$. The proof of knowledge is then constructed as follows:

1. **Commitments**

   (a) compute $\bar{W} = (\bar{w}_1, \ldots, \bar{w}_m)$ with $\bar{w}_\alpha = 0 \quad \text{and} \quad \bar{w}_i \in_R \mathbb{Z}_q$ for $i \neq \alpha$

   (b) assign to $\bar{V} = (\bar{v}_{1.0\ldots,1}, \ldots, \bar{v}_{m.0\ldots,.})$ a random tuple satisfying $E|_{W=\bar{W}}$

   (c) assign to each node n in the forest $\tilde{F}$ a commitment T_n in the following way:

   - if n is a leaf of type $\mathrm{DL}(g, y)$ in the tree $\tilde{F}_i$ then

   $$T_n = (y^{\bar{w}_i} g^{\bar{v}_n})$$

   - if n is a leaf of type $\mathrm{REP}((g_1, \ldots, g_k), y)$ in the tree $\tilde{F}_i$ then

   $$T_n = (y^{\bar{w}_i} \prod_{j=1}^{k} g_j^{\bar{v}_{n,j}})$$

   - if n is a leaf of type $\mathrm{LE}((a_1, \ldots, a_k), b)$ then T_n is the empty tuple ()
   - if n is an inner node of type $\otimes$ or $\cap$ then

   $$T_n = T_{n\|0} \circ T_{n\|1}$$

   The commitment T is then computed as

   $$T = T_{1.0} \circ \ldots \circ T_{m.0}$$

2. **Challenge**

   The challenge $C = (c_1, \ldots, c_m)$ is computed as follows

   $$c_i = \begin{cases} \mathcal{H}(\tilde{F}, T) - \sum_{j=1}^{m} \bar{w}_j \pmod q & \text{for } i = \alpha \\ \bar{w}_i & \text{otherwise} \end{cases}$$

3. **Response**

   Given $K \in \tilde{F}_\alpha$ the prover can construct a tuple X satisfying the following conditions (the components of X are labeled in the same way as the components of V):

- $x_{n,j} = 0$ for all indices j if the leave n is *not*in the tree $\tilde{F}_\alpha$
- if n is a leaf of the type DL or REP in $\tilde{F}_\alpha$ then the sub-tuple $(x_{n,1},...,x_{n,k})$ is an element of the set defined by the type of the leaf.
- $X_{\alpha.0}$ satisfies the equations $E_{\alpha.0}|_{w_\alpha=-1}$ (where $X_{\alpha.0}$ is the sub-tuple of X corresponding to the sub-tuple $V_{\alpha.0}$ of V)

The response $R = (r_{1.0...,1},...,r_{m.0...,.})$ is then defined by

$$r_{n,j} = \bar{v}_{n,j} - c_\alpha x_{n,j} \pmod q$$

for all leaves n and all indices j.

The proof of knowledge is the pair (C, R).

Verifying a proof

The verification of a proof (C, R) consists of the following two steps:

1. **Reconstructing the commitment** by assigning to each node n in the forest $\tilde{F}$ a tuple T'_n in the following way:
   - if n is a leaf of type DL(g, y) in the tree $\tilde{F}_i$ then
   $$T'_n = (y^{c_i} g^{r_n})$$
   - if n is a leaf of type REP$((g_1,...,g_k), y)$ in the tree $\tilde{F}_i$ then
   $$T'_n = (y^{c_i} \prod_{j=1}^{k} g_j^{r_{n,j}})$$
   - if n is a leaf of type LE$((a_1,...,a_k), b)$ then T'_n is the empty tuple ()
   - if n is an inner node of type $\otimes$ or $\cap$ then
   $$T'_n = T'_{n\|0} \circ T'_{n\|1}$$

   The reconstructed commitment T' is then

   $$T' = T'_{1.0} \circ ... \circ T'_{m.0} \ .$$

2. **Verifying the challenge and the response** by
   - verifying that $\mathcal{H}(\tilde{F}, T') = \sum_{i=1}^{m} c_i \pmod q$ and by
   - verifying that R satisfies $E|_{W=C}$

6 Example

The following example should clarify the method presented in Section 5.

Example 4. Assume that the prover knows x_1, x_2, and x_3 such that

$$z = h^{x_1}, \quad y = g_1^{x_2} g_2^{x_3}, \quad \text{and} \quad b = a_1x_1 + a_2x_2 + a_3x_3 \pmod q$$

and wants to prove the knowledge of the discrete logarithm of z to the base h and the representation of y to the bases g_1 and g_2. Furthermore, he or she wants to prove that either $b = a_1x_1 + a_2x_2 + a_3x_3 \pmod q$ or $b = a_1x_2 + a_2x_3 + a_3x_1 \pmod q$ holds (without giving further information on x_1, x_2, and x_3, of course).

The knowledge specification for this proof is

$$F = \Big(\big(\mathrm{DL}(h,z) \otimes \mathrm{REP}((g_1,g_2),y)\big) \cup \big(\mathrm{REP}((g_1,g_2),y) \otimes \mathrm{DL}(h,z)\big)\Big) \cap \mathrm{LE}((a_1,a_2,a_3),b);$$

In order to construct the proof system this specification must first be transformed into the tree-representation which is achieved by applying transformation III once:

$$\begin{aligned}\tilde{F} = \Big(\big(\mathrm{DL}(h,z) \otimes \mathrm{REP}((g_1,g_2),y)\big) \cap \mathrm{LE}((a_1,a_2,a_3),b)\Big) \cup \\ \Big(\big(\mathrm{REP}((g_1,g_2),y) \otimes \mathrm{DL}(h,z)\big) \cap \mathrm{LE}((a_1,a_2,a_3),b)\Big).\end{aligned}$$

This formula is depicted in Figure 1.

Next the prover has to built the lists of variables and the set of equations for each node. Here we do this only for the tree $\tilde{F}_1$, the lists and sets for the tree $\tilde{F}_2$ look similar.

$$\begin{array}{ll}
\text{node 1.000:} & V_{1.000} = (v_{1.000,1}) \\
& E_{1.000} = \emptyset \\
\text{node 1.001:} & V_{1.001} = (v_{1.001,1}, v_{1.001,2}) \\
& E_{1.001} = \emptyset \\
\text{node 1.00:} & V_{1.00} = V_{1.000} \circ V_{1.001} = (v_{1.000,1}, v_{1.001,1},\, v_{1.001,2}) \\
& E_{1.00} = E_{1.000} \cup E_{1.001} = \emptyset \\
\text{node 1.01:} & V_{1.01} = (v_{1.01,1}, v_{1.01,2}, v_{1.01,2}) \\
& E_{1.01} = \{a_1v_{1.01,1} + a_2v_{1.01,2} + a_3v_{1.01,2} = -w_1b\} \\
\text{node 1.0:} & V_{1.0} = (v_{1.000,1}, v_{1.001,1}, v_{1.001,2}, v_{1.01,1}, v_{1.01,2}, v_{1.01,2}) \\
& E_{1.0} = \{v_{1.01,1} = v_{1.000,1}, v_{1.01,2} = v_{1.001,1}, v_{1.01,2} = v_{1.001,2}, \\
& \qquad\qquad a_1v_{1.01,1} + a_2v_{1.01,2} + a_3v_{1.01,2} = -w_1b\}
\end{array}$$

Finally the sets $E_{1.0}$ and $E_{2.0}$ are merged and the prover obtains E which is the set of the following equations over $\mathbb{Z}_q$:

$$\begin{aligned}E = \{\ & v_{1.01,1} = v_{1.000,1}, v_{1.01,2} = v_{1.001,1}, v_{1.01,2} = v_{1.001,2}, a_1v_{1.01,1} + a_2v_{1.01,2} + a_3v_{1.01,2} = -w_1b, \\ & v_{2.01,1} = v_{2.000,1}, v_{2.01,2} = v_{2.000,2}, v_{2.01,2} = v_{2.001,1}, a_1v_{2.01,1} + a_2v_{2.01,2} + a_3v_{2.01,2} = -w_2b\}\end{aligned}$$

and after also assigning $V = V_{1.0} \circ V_{2.0}$, and $W = (w_1, w_2)$ the prover is able to construct the proof. He chooses $\bar{W} = (\bar{w}_1, \bar{w}_2)$ as $(0, w)$ for some $w \in_R \mathbb{Z}_q$ and a random tuple $\bar{V} \in_R \mathbb{Z}_q^{12}$ satisfying the equations $E|_{W=\bar{W}}$. This can be achieved by randomly choosing $\bar{v}_1,\ldots,\bar{v}_6$ in $\mathbb{Z}_q$ such that the equations

$$\begin{aligned} a_1\bar{v}_1 + a_2\bar{v}_2 + a_3\bar{v}_3 &= 0 \pmod q \\ a_1\bar{v}_4 + a_2\bar{v}_5 + a_3\bar{v}_6 &= -wb \pmod q \end{aligned}$$

hold and then setting

$$V = (\bar{v}_1, \bar{v}_2, \bar{v}_3, \bar{v}_1, \bar{v}_2, \bar{v}_3, \bar{v}_4, \bar{v}_5, \bar{v}_6, \bar{v}_4, \bar{v}_5, \bar{v}_6).$$

The commitments for the nodes can now be computed:

$$\begin{array}{ll}
T_{1.000} = (h^{\bar{v}_1}) & T_{2.000} = (z^w h^{\bar{v}_1}) \\
T_{1.001} = (g_1^{\bar{v}_2} g_2^{\bar{v}_3}) & T_{2.001} = (y^w g_1^{\bar{v}_2} g_2^{\bar{v}_3}) \\
T_{1.00} = (h^{\bar{v}_1}, g_1^{\bar{v}_2} g_2^{\bar{v}_3}) & T_{2.00} = (z^w h^{\bar{v}_1}, y^w g_1^{\bar{v}_2} g_2^{\bar{v}_3}) \\
T_{1.01} = () & T_{2.01} = () \\
T_{1.0} = (h^{\bar{v}_1}, g_1^{\bar{v}_2} g_2^{\bar{v}_3}) & T_{2.0} = (z^w h^{\bar{v}_1}, y^w g_1^{\bar{v}_2} g_2^{\bar{v}_3})
\end{array}$$

Using $T = T_{1.0} \circ T_{2.0}$ the prover can compute the challenge

$$C = (c_1, c_2) = (\mathcal{H}(\tilde{F}, T) - w \pmod q, w).$$

To calculate the response R the prover builds the list $X = (x_1, x_2, x_3, x_1, x_2, x_3, 0, 0, 0, 0, 0, 0)$ (note that $\alpha = 1$) and computes the components $r_{i.j,\ell}$ of R as (all equations are modulo q, components listed in the right order)

$$\begin{array}{llll}
\text{node } 1.000: & r_{1.000,1} = \bar{v}_1 - c_1 x_1 & \text{node } 2.000: & r_{2.000,1} = \bar{v}_4, \\
\text{node } 1.001: & r_{1.001,1} = \bar{v}_2 - c_1 x_2, & & r_{2.000,2} = \bar{v}_5 \\
& r_{1.001,2} = \bar{v}_3 - c_1 x_3 & \text{node } 2.001: & r_{2.001,1} = \bar{v}_6 \\
\text{node } 1.01: & r_{1.01,1} = \bar{v}_1 - c_1 x_1, & \text{node } 2.01: & r_{2.01,1} = \bar{v}_4, \\
& r_{2.01,2} = \bar{v}_2 - c_1 x_2, & & r_{2.01,2} = \bar{v}_5, \\
& r_{2.01,3} = \bar{v}_3 - c_1 x_3 & & r_{2.01,3} = \bar{v}_6
\end{array}$$

The resulting proof is the pair (C, R).

Let us now see how a verifier proceeds to check the validity of the proof. As the first step the verifier must reconstruct the commitment by assigning to each node in $\tilde{F}_1$ and $\tilde{F}_2$ a tuple T'_n:

$$\begin{array}{ll}
T'_{1.000} = (z^{c_1} h^{r_{1.000,1}}) & T'_{2.000} = (y^{c_2} g_1^{r_{2.000,1}} g_2^{r_{2.000,2}}) \\
T'_{1.001} = (y^{c_1} g_1^{r_{1.001,1}} g_2^{r_{1.001,2}}) & T'_{2.001} = (z^{c_2} h^{r_{2.001,1}}) \\
T'_{1.00} = (z^{c_1} h^{r_{1.000,1}}, y^{c_1} g_1^{r_{1.001,1}} g_2^{r_{1.001,2}}) & T'_{2.00} = (y^{c_2} g_1^{r_{2.000,1}} g_2^{r_{2.000,2}}, z^{c_2} h^{r_{2.001,1}}) \\
T'_{1.01} = () & T'_{2.01} = () \\
T'_{1.0} = (z^{c_1} h^{r_{1.000,1}}, y^{c_1} g_1^{r_{1.001,1}} g_2^{r_{1.001,2}}) & T'_{2.0} = (y^{c_2} g_1^{r_{2.000,1}} g_2^{r_{2.000,2}}, z^{c_2} h^{r_{2.001,1}})
\end{array}$$

and gets $T' = T'_{1.0} \circ T'_{2.0}$. Then the verifier checks the challenge and the equations of $E|_{W=C}$ (again, all equations are modulo q):

$$\begin{array}{lll}
\mathcal{H}(\tilde{F}, T') = c_1 + c_2 & r_{1.000,1} = r_{1.01,1} & r_{2.000,1} = r_{2.01,1} \\
a_1 r_{1.01,1} + a_2 r_{1.01,2} + a_3 r_{1.01,3} = -c_1 b & r_{1.001,1} = r_{1.01,2} & r_{2.000,2} = r_{2.01,2} \\
a_1 r_{2.01,1} + a_2 r_{2.01,2} + a_3 r_{2.01,3} = -c_2 b & r_{1.001,2} = r_{1.01,3} & r_{2.001,1} = r_{2.01,3}
\end{array}$$

It can easily be seen that these equations hold if the proof is constructed as described above.

7 Extensions and open problems

In this report we have shown how to construct complex proofs of knowledge. In order to keep the notation and the method for deriving proofs as simple as possible, several possible extensions and optimizations have been omitted. .

Some of these extensions and improvements are quite obvious. For instance, instead of returning the whole tuple R which must satisfy the equations in $E|_{W=C}$, the prover can send only as many components of R as are sufficient to compute the other components using the linear equations in $E|_{W=C}$. In Example 4 it suffices to return the values $r_{1.01,1}$, $r_{1.01,1}$, $r_{2.01,1}$, and $r_{2.01,1}$; all other eight components of R can then easily be computed from the linear equations.

Another simple extension is to combine proofs of knowledge of discrete logarithms in different groups, or even combinations of proofs about different problems. However, one should be careful not to intersect sets of different types since this could result in misinterpretations of the proofs.

Finally, using techniques from [17, 16], proofs can also be blindly issued, meaning that the prover helps a recipient to obtain a valid proof of knowledge without obtaining information about it. An important application of such "blindly issued" proofs are anonymity protecting digital payment systems (e.g. see [1, 2, 5]).

An interesting open problem is the design of efficient proofs of knowledge combined with non-linear equations, such as a proof that one discrete logarithm equals the third power of another discrete logarithm (there are a few non-linear relations that can be efficiently proved, such as proving that one discrete logarithm is the square or the inverse of another one, but it seems difficult to generalize these methods).

Acknowledgments

The first author is supported by the Swiss Federal Commission for Technology and Innovation (KTI) and by the Union Bank of Switzerland.

References

[1] S. Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, CWI, Apr. 1993.

[2] S. Brands. Untraceable off-line cash in wallets with observers. manuscript, CWI, 1993.

[3] G. Brassard, C. Crépeau, and M. Yung. Everything in NP can be argued in perfect zero-knowledge in a bounded number of rounds. In G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, editors, *Proceedings of 16th ICALP*, volume 372 of *Lecture Notes in Computer Science*, pages 123–136. Springer-Verlag, 1989.

[4] J. Camenisch, U. Maurer, and M. Stadler. Digital payment systems with passive anonymity-revoking trustees. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Computer Security — ESORICS 96*, volume 1146 of *Lecture Notes in Computer Science*, pages 33–43. Springer Verlag, 1996.

[5] J. Camenisch, J.-M. Piveteau, and M. Stadler. An efficient fair payment system. In *3rd ACM Conference on Computer and Communicatons Security*, pages 88–94, New Delhi, Mar. 1996. Association for Computing Machinery.

[6] D. Chaum, J.-H. Evertse, and J. van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In D. Chaum and W. L. Price, editors, *Advances in cryptology — EUROCRYPT '87*, volume 304 of *Lecture Notes in Computer Science*, pages 127–141. Springer-Verlag, 1988.

[7] D. Chaum, J.-H. Evertse, J. van de Graaf, and R. Peralta. Demonstrating possession of a discrete logarithm without revealing it. In A. M. Odlyzko, editor, *Advances in Cryptology — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 200–212. Springer-Verlag, 1987.

[8] D. Chaum and T. Pedersen. Transferred cash grows in size. In R. A. Rueppel, editor, *Advances in Cryptology — EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*, pages 390–407. Springer-Verlag, 1993.

[9] D. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer-Verlag, 1993.

[10] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y. G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer Verlag, 1994.

[11] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. Multi-authority secret-ballot elections with linear work. In U. Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83. Springer Verlag, 1996.

[12] A. De Santis, G. Di Crescenzo, G. Persiano, and M. Yung. On monotone formula closure of SZK. In *35th Annual Symposium on Foundations of Computer Science*, pages 454–465, Santa Fe, New Mexico, 20–22 Nov. 1994. IEEE.

[13] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1:77–94, 1988.

[14] O. Goldreich, S. Micali, and A. Wigderson. How to prove all NP statements in zero-knowledge and a methodology of cryptographic protocol design. In A. M. Odlyzko, editor, *Advances in Cryptology — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 171–185. Springer-Verlag, 1987.

[15] L. C. Guillou and J.-J. Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In C. G. Günther, editor, *Advances in Cryptology — EUROCRYPT '88*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128. Springer Verlag, 1988.

[16] T. Okamoto. Provable secure and practical identification schemes and corresponding signature schemes. In E. F. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer-Verlag, 1993.

[17] T. Okamoto and K. Ohta. Disposable zero-knowledge authentications and their applications to untraceable electronic cash. In G. Brassard, editor, *Advances in Cryptology — CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 481–496. Springer-Verlag, 1990.

[18] C. P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.