RADBOUD UNIVERSITY NIJMEGEN

MASTER THESIS

Efficient Delegation of Idemix Credentials

Author:
Manu DRIJVERS
s3040429

Supervisor:
Jaap-Henk HOEPMAN

Second Reader:
Bart JACOBS

June 25, 2014

Abstract

Attribute-based credential systems like Idemix are a form of identity management that protects the user's privacy. The user has credentials containing his own attributes, which he can selectively and unlinkably disclose to a verifier.

In many scenarios one would like to delegate tasks and authority, e.g. a financial director might want to delegate the task and authority to file company taxes to a subordinate. Currently Idemix does not support delegation of credentials, and can therefore not be used in scenarios where delegation is required.

We extend Idemix to allow one step of delegation of credentials. This means one can delegate his credentials to a delegatee, but the delegatee cannot delegate this credential any further. Our construction is efficient, showing a delegated credential takes 24 modular exponentiations. In order to achieve this, we introduce RSA signatures with a half-domain hash, RSA-HDH, which we prove secure.

Contents

Chapter 1

Introduction

With technological progress, we've seen more and more services move from the physical to the digital world. For users this is very convenient, but it also endangers their privacy. In the digital world the user are often required to register some personal information, which allows tracking of their behavior. Often this information is identifying, which allows multiple services to combine their user profiles and learn more about their customers. All this information about the user is stored at the services, outside of control of the user.

Users should therefore be very careful about the personal information they disclose. Attribute-based credential systems like Idemix can protect the user's privacy by limiting the information disclosed and by hiding usage patterns. The user is in charge of his own identity, and decides when and where to disclose personal information. In this thesis, we will investigate ways to efficiently delegate Idemix credentials, such that we can use this privacy protecting technique in more scenarios. First, a short introduction to identity management and attribute-based credentials is given, after which we will state the problem and motivate the need for this research.

1.1 Identity Management

To use services, users are often required to register first. The domain that offers the service wants to know some attributes of the users. For example, a prospective student must first register at the university (the domain), in which the user will tell his name and address (attributes). We will refer to such a collection of attributes at a domain as an identity. One person will have many different identities, as he uses services of many different domains.

Identity Management (IdM) governs the processes of creating, managing and using digital identities. Three parties are involved in an IdM system: users, relying parties (RP), and identity providers (IdP). The user wants to use a service offered by a RP, and the RP relies on the IdP to verify identities from users. When a user requests a service at a RP, he will have to authenticate at the IdP. In case of a network-based IdM system, the IdP will give the user a token, which the user shows to the RP proving that he has authenticated. The RP may request more information on the user's identity from the IdP directly. In a claim-based IdM system, before authentication the RP states what information about the user's identity he wants.

The user authenticates at the IdP and the IdP returns so-called claims, which contain the requested information, which the user can forward to the RP.

Over the years, IdM has become more complex. Users use more and more digital services and consequently gain more identities at different domains, for which they often have to remember their username and password. Also, domains want to allow identities from other domains. For example, university A now also wants to let members of university B access their digital library. This introduced Federated Identity Management (FIdM), in which a single identity is used in multiple domains, the circle of trust, as shown in Figure 1.1. This is very popular, partly due to the ease of use: a user authenticates at a IdP once, and can then use services from all RPs in the circle of trust.

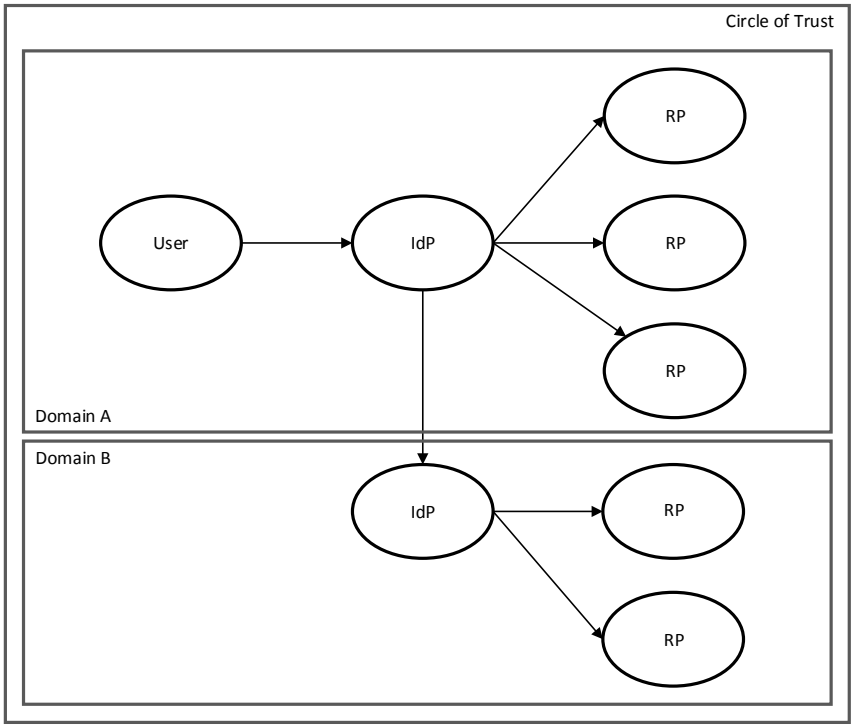


Figure 1.1: Federated Identity Management

Federated IdM has its downsides. Users must put a great amount of trust in the IdP, as he knows everything about them. The IdP can impersonate any user, and is an interesting target for data theft, because of all the valuable data it has. The IdP is also capable of monitoring all actions of the users, which is a violation of the user's privacy. In some instantiations of Federated IdP (e.g. DigiD, the identity management system by the Dutch government), the RPs will also be able to identify the user, as the IdP gives a unique identifier of the user. This allows RPs to track a user's behavior. An overview of current problems in IdM is presented in [AHS13].

1.2 Attribute-Based Credentials

Attribute-based credentials (also known as pseudonym systems and anonymous credentials) take a different approach to IdM. The concept has been introduced by Chaum [Cha85], and many improved constructions have been introduced since [CE87, Dam90, LRSW00, Bra99, CL01, CL03, CL04]. This section explains which problems attribute-based credentials solve, followed by a high-level overview of how these systems work.

1.2.1 Goal of ABCs

In many everyday scenarios you have to authenticate before you are authorized to perform an action. Traditionally this involves identification: revealing your identity. When buying alcohol at a liquor store, the customer must prove that he is of age. To do so, he must show his passport. This passport indeed shows whether the customer is of age or not, but also reveals more information: it contains a full name, social security number, and document number. All of this information is irrelevant to the storekeeper, and violates the privacy of the customer. Even the date of birth should not be disclosed, as this tells the shopkeeper the exact age of the customer, whereas all he needs to know is whether his age is over a certain limit.

The second violation of privacy is linkability. Because the shopkeeper can uniquely identify the customer (by e.g. his name, social security number, or passport number), he will notice when the same customer returns. This allows the shopkeeper to keep track of what you buy. Even worse, the shopkeeper could collude with other shopkeepers and combine their profiles. Since the customer is uniquely identified, his actions in different stores can be linked which gives the shopkeepers more information about the customer.

Attribute-based credentials (ABCs) allow users to authenticate without identification. Instead, a user can directly prove that he has certain attributes. Attributes are pieces of information about the owner, such as “is of age over 18”, “my nationality is ...”, or “my name is ...”. Note that some attributes will identify you (in this example, your name), whereas others do not (over 18 and nationality). Let us revisit the liquor store example. In an ABC setting, the customer proves he has the “is of age over 18” attribute. The first problem (revealing too much information) does not occur, as the customer shows exactly what he needs to. The second problem is also mitigated. Because the user only shows a non-identifying attribute, the shopkeeper cannot recognize the same customer returning, and therefore cannot build user profiles.

1.2.2 How ABCs work

How can a user prove that he has some attribute, without revealing anything else? This is achieved by using credentials. A set of attributes is signed by an issuer. The issuer must be an authority on the subject of the attributes, as depicted in Figure 1.2. For example, the university could issue credentials containing attributes “is a student”, “has student ID 1234”, “studies computer science”, at level “MSc”. Suppose this student now visits a museum and wants to get receive a student discount. He can now show the credential issued by the university, which states that he is a student. Since this has been signed by the university, the museum is convinced that he indeed is a student.

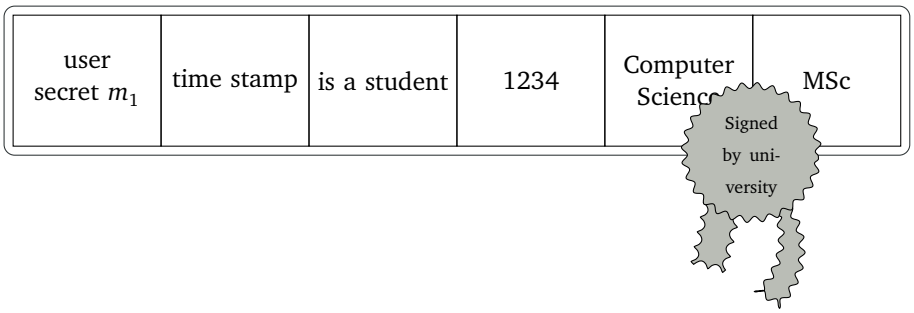


Figure 1.2: Graphical representation of a student credential[1]

An important property of attribute-based credential systems is selective disclosure, which allows a user to choose which attributes of a credential are revealed to a relying party. The student that wants the student discount only needs to prove that he is a student. Selective disclosure allows him to prove just that, and hide the other attributes. Another important property is multi-show unlinkability. When the student returns to the museum and again shows his student credential, the museum should not be able to tell that this is the same student. Finally we need issuer-unlinkability, which ensures that an issuer cannot link the issuance of a credential to the showing of that credential.

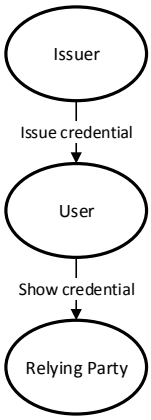


Figure 1.3: IdM using attribute-based credentials

Compared to other forms of IdM, the role of IdP is replaced by credential issuers. As shown in Figure 1.3, this form of IdM places the user in the center: issuers talk to users, and users talk to RPs, but issuers never talk to RPs. This has many advantages. The security risks of a corrupt IdP that uses one of the identities he has stored is treated, as there no longer is an IdP with stored identities, the user stores his own identities in the form of credentials. This also removes the risk of data theft at an IdP. This setup also helps protect the user's privacy. A credential issuer cannot keep

[1] Figure by Tim van de Kamp, used with permission

track of where credentials are being used, so he cannot monitor the behavior of users.

1.3 Problem Statement

In some scenarios, one would like to delegate authority or privileges. This is currently not possible with Idemix credentials. In this thesis, we want to answer the following question: "Can we extend Idemix to allow efficient delegation of credentials?" To answer this question, we answer the following subquestions:

1. What zero-knowledge proof techniques are available?
2. How does Idemix work?
3. What are the requirements on delegation of attribute-based credentials?
4. Which constructions of credential delegation are already available?

An overview of zero-knowledge proof techniques is provided in Chapter 3, followed by a cryptographic overview of Idemix in Chapter 4. In Chapter 5 we set the requirements on delegatable credentials. In this chapter we also answer the main question and propose a way to delegate Idemix credentials. Chapter 6 provides an overview of other constructions of delegatable credentials.

1.4 Motivation

We now motivate why we need delegation of attribute-based credentials, why we want to delegate Idemix credentials, and why it must be efficient.

Delegation of tasks and authority happens every day. In software development, the lead developer that must sign-off all the code might authorize some senior developer to sign-off code in one specific branch of the project. We imagine a privacy friendly world where ABCs are used for many tasks, including the example of signing-off code. The lead developer has attributes that allow him to sign-off code for the entire project. He could delegate the task of signing-off code for a project branch to a senior developer by delegating a credential. Another example is filing taxes for a company. Suppose the financial officer is responsible for filing the company's taxes. He might delegate this task to a subordinate, and by delegating his credential, give this subordinate the authority required to do so.

Delegation is not only useful for person-to-person delegation. Suppose a user has Idemix on a smart card that he uses to access his email. He will have to insert his smart card into his laptop such that the email client can fetch the new messages. The user however wants his laptop to fetch email messages also when he is not using his laptop and his smart card is not inserted. We can image that the laptop is an individual Idemix user (e.g. using an extended trusted platform module that supports Idemix). The user could solve the problem by delegating his email access credential to the laptop.

Privacy could also be increased using delegatable credentials. Driver's licenses are often handled by some national authority. This authority has local offices that issue driver's licenses. When using ABCs for driver's licenses, the public key of the local office still reveals which office issued the driver's license and therefore reveals

roughly where the owner lives. Using delegation, the national authority could issue credentials that allow issuance of driver's licenses. Every local office has such a credential. To issue a driver's license to a person, the local office delegates the credential to this person. Now all driver's licenses will be a delegated credential of a credential issued by the national authority, that does not reveal which local office was involved.

Another area that could benefit from delegatable attribute-based credentials is the health care system, explained in detail in [CL11]. Health insurers issue credentials to their customers, which allows them to prove they have health insurance. After seeing a doctor, the patient delegates this credential to the doctor, indicating that he has received a certain form of health care from this doctor. The doctor bills the insurance company, showing the delegated credential, proving that one of their customers indeed received health care. One of the advantages of this approach is that health insurers do not know which of their customers received which form of health care.

There are constructions of ABCs specifically designed to allow delegation of credentials [CL06, BCC+09, CKLM13]. However, these constructions heavily rely on computationally expensive bilinear pairings. Also, at this point these systems are unused. Two ABC systems are being used: IBM's Idemix [IBM13] and Microsoft's U-Prove [Bra99, Paq13]. The main difference is that Idemix offers multi-show unlinkability, whereas U-Prove does not. We therefore prefer Idemix.

We always want cryptographic systems to be efficient, but the IRMA project is extra motivation to create an efficient solution.

1.4.1 IRMA project

The IRMA project[2], short for "I Reveal My Attributes", aims to put Idemix into practice by creating an efficient smart card implementation of Idemix [VA13]. A smart card implementation can bring extra security as the key material is stored securely, while it can also improve usability, as showing a smart card is very easy. Delegation of Idemix credentials could also contribute to the IRMA project, but only if the construction is efficient enough to run on smart cards.

[2] see `www.irmacard.org` for more information

Chapter 2

Cryptographic Preliminaries

In this chapter we introduce the required cryptographic preliminaries.

2.1 Notation

Let $\mathbb{G}$ be a multiplicative cyclic group. We use $\langle g \rangle$ to denote the set of elements generated by applying the group operation repeatedly on g: $\{g, g \cdot g = g^2, g^3, ...\}$. If this results in the entire group $\mathbb{G}$, i.e. $\langle g \rangle = \mathbb{G}$, we call g a generator of $\mathbb{G}$.

Checking whether an equality holds will be denoted using $a \stackrel{?}{=} b$, which checks whether a is equal to b.

A group of Quadratic Residues modulo n will be denoted by $QR(n)$, which is equal to $\{q \mid \exists_x x^2 = q \pmod{n}\}$. We use $\{0,1\}^l$ to denote the set of integers $\{0, ..., 2^l - 1\}$, and $\pm\{0,1\}^l$ the set $\{-2^l + 1, ..., 2^l - 1\}$.

2.2 Probability Theory

Let V be a finite set. We write $x \in_R V$ to denote that x is taken uniformly at random from V: $Pr[x = v] = \frac{1}{|V|}$ for all $v \in V$. We use the statistical distance Δ to compare two random variables. The statistical distance between random variables X and Y with set of possible values V is defined as $\Delta(X, Y) = \frac{1}{2} \sum_{v \in V} |Pr[X = v] - Pr[Y = v]|$, so two equally distributed random variables will have a statistical distance of 0, and two random variables that share no possible outcome have a distance of 1.

2.3 Indistinguishability

A function $f : \mathbb{N} \to \mathbb{R}$ is called negligible if for every positive polynomial p there is a constant $c \in \mathbb{N}$ such that for all $k \geq c$ we have $f(k) \leq \frac{1}{p(k)}$. We denote this by $f(k) \cong 0$.

Let $\{X_i\}_{i \in I}$ and $\{Y_i\}_{i \in I}$ be families of random variables with $|X_i|$ and $|Y_i|$ of size polynomial in $|i|$. We have the following notions of indistinguishability:

Perfect Indistinguishable when the random variable families are equally distributed: $\Delta(X_i, Y_i) = 0$.

Statistically Indistinguishable when they are almost equally distributed: $\Delta(X_i, Y_i) \cong 0$.

Computationally Indistinguishable if probabilistic polynomial time (PPT) algorithm has non-negligible success at distinguishing the two random variables: for all PPT-distinguishers $\mathcal{D}$ the advantage $Adv_{\mathcal{D}}(X_i, Y_i) \cong 0$, with $Adv_{\mathcal{D}}(X_i, Y_i) = |Pr[\mathcal{D}(X_i) = 1] - Pr[\mathcal{D}(Y_i) = 1]|$.

2.4 Bilinear Pairings

Suppose we have additive groups $\mathbb{G}_1$, $\mathbb{G}_2$, and multiplicative group $\mathbb{G}_T$. A pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a function such that we have

Bilinear $e(aP, bQ) = e(P, Q)^{ab}$

Non-degenerate $\langle P \rangle = \mathbb{G}_1 \wedge \langle Q \rangle = \mathbb{G}_2 \implies \langle e(P, Q) \rangle = \mathbb{G}_T$

Generally three types of pairings are considered. The first has $\mathbb{G}_1 = \mathbb{G}_2$. The second type $\mathbb{G}_1 \neq \mathbb{G}_2$, but there is an isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$. The third type also has $\mathbb{G}_1 \neq \mathbb{G}_2$ but there is no such isomorphism. There are no known pairings such that $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}_T$.

The non-degeneracy requirement prevents a pairing that maps all elements to a single element in $\mathbb{G}_T$. $\mathbb{G}_1$ and $\mathbb{G}_2$ are often called the gap groups, and $\mathbb{G}_T$ the target group. In practice, $\mathbb{G}_1$ and $\mathbb{G}_2$ are typically subgroups of elliptic curves over $\mathbb{F}_p^*$, and $\mathbb{G}_T$ is an extension field of $\mathbb{F}_p^*$. Since elliptic curve groups are usually written additively, we choose to write the gap groups additively as well.

In type 1 pairings, it is easy to solve the DDH-problem in the gap group: given g, g^x, g^y, g^z, we can decide $z \stackrel{?}{=} ab$ by checking $e(g^x, g^y) \stackrel{?}{=} e(g, g^z)$. This demonstrates that groups with bilinear pairings may require different complexity assumptions.

2.5 Complexity Assumptions

The following problems are often assumed to be hard. This means that we assume an attacker cannot compute certain things. Under these assumptions we can create secure cryptographic primitives. Table 2.1 shows the names of those assumptions and the corresponding problems.

Given a multiplicative group $\mathbb{G}$ generated by g:

Discrete Log Problem Given g^x, output x.

Diffie-Hellman Problem Given g^a and g^b, output g^{ab}.

Decisional Diffie-Hellman Problem Given g^a, g^b, g^z, output whether $ab \stackrel{?}{=} z$ holds.

Formal definitions of the DH and DDH problems can be found in [Bon98].

Given n, the product two large primes p, q:

RSA Problem Given e and c, output the e-th root $m = c^{\frac{1}{e}}$.

Flexible RSA Problem Given c, output any $e > 1$ and the e-th root $m = c^{\frac{1}{e}}$

Assumption	Problem
DL	Discrete Logarithm Problem
DH	Diffie-Hellman Problem
DDH	Decisional Diffie-Hellman Problem
RSA	RSA Problem
Strong RSA	Flexible RSA Problem
DLIN	Decision Linear Problem
BDDH	Bilinear Decisional Diffie-Hellman Problem
qSDH	q-Strong Diffie-Hellman Problem
SD	Subgroup Decision Problem
SXDH	Symmetric External Diffie-Hellman Problem

Table 2.1: Names of complexity assumptions and the corresponding problems assumed to be hard

Given $\langle g_1 \rangle = \mathbb{G}_1$, $\langle g_2 \rangle = \mathbb{G}_2$ of prime order p and bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$:

Decision Linear Problem given arbitrary generators u, v, h of $\mathbb{G}_1$, and u^a, v^b, h^c, output whether $a + b \stackrel{?}{=} c$. This problem is introduced in [BBS04].

q-Strong Diffie-Hellman Problem Given $(g_1, g_2, {g_2}^x, ..., {g_2}^{x^q}$, output a pair $(c, {g_1}^{1/(x+c)})$ with $c \in \mathbb{Z}_p^*$. This problem is introduced in [BB08].

Symmetric External Diffie-Hellman Problem Solve the DDH problem in $\mathbb{G}_1$ or $\mathbb{G}_2$. A formal definition is given in [ACHdM05].

Given $\mathbb{G}$ of composite order n with secret factorization $n = pq$, G_T and pairing e:

Subgroup Decision Problem Given an element $u \in \mathbb{G}$, output whether $\langle u \rangle$ has order n or q. This problem is introduced in [BGN05].

2.6 Commitment Scheme

Alice and Bob like to play the game battleship. In the beginning of this game, both players must commit to the arrangement of their ships. They do so by writing this arrangement down on a piece of paper, and fold the paper. An example of this is shown in Figure 2.1. The outside of this paper does not reveal the arrangement of the ships, it is *hiding*. But by writing the locations down, Alice and Bob cannot change the locations during the game, it is *binding*. At the end of the game, both players will open their folded paper and they can verify that the other player did not cheat. Now suppose Alice and Bob want to play battleship over email. To do so, they need a digital version of the piece of paper: a cryptographic commitment scheme.

More generally, a cryptographic commitment scheme allows one to choose a value without revealing it, yet such that everyone is ensured that the value will not be changed later. This concept is introduced in [Blu83]. A commitment scheme defines a function *Commit*. Such a scheme consists of two rounds:

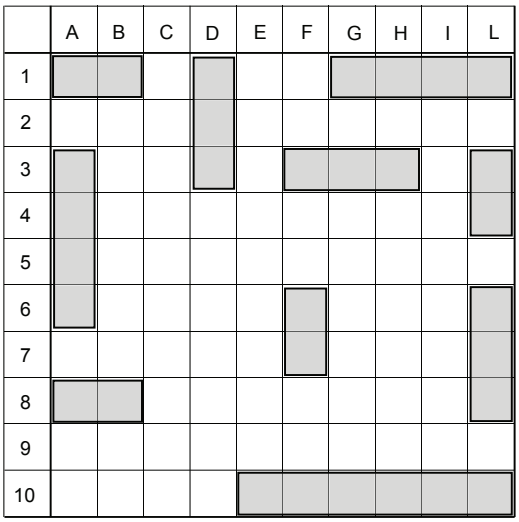


Figure 2.1: A battleship arrangement of ships. Source: `http://commons.wikimedia.org/wiki/File:Battleship_game_board.svg`

Commit To commit to value x, the committer chooses random u and calculates $c = Commit(x,u)$.

Reveal To reveal that c was a commitment to x, the committer publishes x and the opening u. The commitment can now be checked by $c \stackrel{?}{=} Commit(x,u)$.

A commitment scheme must be *binding* and *hiding*. Binding requires that after committing to $c = Commit(x,u)$, the committer cannot change his mind and open the commitment to $c = Commit(x',u')$ with $x \neq x'$. When this is computationally infeasible, we call the scheme computationally binding. If this is impossible even with unlimited computing power (i.e. $\neg\exists u,x,u',x' : x \neq x' \wedge Commit(x,u) = Commit(x',u')$), the scheme is called information-theoretically binding.

A commitment scheme is hiding if one cannot extract x from $Commit(x,u)$ with non-negligible probability. When this is computationally infeasible, the scheme is computationally hiding. If commitments to x and x' are statistically indistinguishable for all x,x', the scheme is information-theoretically hiding.

Note that a scheme cannot be be both information-theoretically binding and information-theoretically hiding. Suppose that the scheme is information-theoretically hiding. This means that commitments to x are statistically indistinguishable from commitments to x'. This can only hold if there exist u and u' such that $Commit(x,u) = Commit(x',u')$, but this prevents information-theoretical binding. Therefore the best we can have is a scheme that is information-theoretically binding and computationally hiding, or one that is computationally binding and information-theoretically hiding.

2.6.1 Pedersen commitment

A Pedersen commitment [Ped92] is a commitment scheme that is information-theoretically hiding and computationally binding under the DL assumption. Take p and q large

primes such that q divides $p-1$. Let $\mathbb{G}$ be a subgroup of $\mathbb{Z}_p^*$ generated by g of order q. Take $h \in_R \mathbb{G} \setminus \{1\}$. Now both g and h are generators of $\mathbb{G}$: the order of $\langle h \rangle$ must be a divisor of q, and because q is prime and $h \neq 1$, the order of $\langle h \rangle$ must be q. To commit to x, a user chooses $u \in_R \mathbb{Z}_q$, and calculates $Commit(x,u) = g^x h^u$.

The commitment is information-theoretically hiding, because for every x, the commitment is a value uniformly distributed over $\mathbb{G}$. It is computationally binding, because suppose one can find x, u, x', u' with $x \neq x' \pmod q$ such that $Commit(x,u) = Commit(x',u')$. This implies that $u \neq u'$. Now $g^{\frac{x-x'}{u'-u}} = h$, so the discrete log of h has been calculated. This contradicts the DL-assumption, so the commitment is computationally binding.

Note that Pedersen does not explicitly state that $h = 1$ must be prevented. However, when $h = 1$, the commitment scheme is not information-theoretically hiding, as a commitment to x will always be g^x, clearly not uniformly distributed over $\mathbb{G}$.

2.7 Digital Signatures

A digital signature allows a signer to sign a message using a secret key, such that anyone can verify the authenticity of the message using the signer's public key. A digital signature scheme consists of three algorithms: KeyGen, Sign, and Verify. The KeyGen algorithm generates the public and secret key. Sign takes as input a message to be signed, and using the secret key is outputs a signature on the message. Verify takes as input a message and a signature, and using the public key it verifies the validity of the signature.

The security of a digital signature is defined by the hardness of creating a signature without the secret key. In [GMR88], categories of attacks on digital signatures and attack results are defined. They define the following attack categories:

Key-Only attack The adversary only knows the signer's public key pk

Known Message attack The adversary is given the public key pk and a number of messages with corresponding signatures $\{m_i, \sigma_i : \text{Verify}_{pk}(m_i, \sigma_i) = 1\}_i$, but the messages are not chosen by the adversary.

Generic Chosen Message attack The adversary may choose a number of messages $\{m_i\}_i$, after which he receives the public key pk and signatures $\{\sigma_i : \text{Verify}_{pk}(m_i, \sigma_i) = 1\}_i$.

Directed Chosen Message attack The adversary is given the public key pk, and may choose a number of messages $\{m_i\}_i$, after which he receives signatures $\{\sigma_i : \text{Verify}_{pk}(m_i, \sigma_i) = 1\}_i$.

Adaptive Chosen Message attack The adversary receives the public key pk and may repeatedly choose a message m_i and receive a signature σ_i with $\text{Verify}_{pk}(m_i, \sigma_i) = 1$. The choice of the next message may depend on the signatures on previous messages.

Note that the adversary is allowed more with every next attack category. Therefore, security for e.g. directed chosen message attacks implies security for known message attacks and key-only attacks.

Attack results are categorized as follows:

Total Break The adversary can derive the signer's secret trap-door information (e.g. his secret key).

Universal Forgery After the attack the adversary can sign any message.

Selective Forgery The adversary can sign any message that he chose before the attack.

Existential Forgery The adversary can find some message-signature pair that is valid and this message has not been signed by the signer. The adversary does not need to have control over which message this is.

Strong existential forgery The adversary can find some message-signature pair that is valid and this pair has not been signed by the signer. The adversary does not need to have control over which message-signature pair this is.

The notion of strong existential forgery has been added by [ADR02], and adds the requirement that a forger cannot forge a new signature on a message (that may previously have been signed). This only differs from existential forgery for signature schemes that allow multiple valid signatures per message. The attack results are given in decreasing order of strength, so every next attack result is implied by the previous result. When using these as security requirements, they are given in increasing degree of security, e.g. if a scheme is universally unforgeable, it is also resistant to a total break. When we say a signature scheme is secure, we often require it to be existentially unforgeable against adaptive chosen message attacks. This guarantees that whenever someone shows a valid signature σ on m, the signer must have signed m before. In some applications a signature that is strongly existentially unforgeable against adaptive chosen message attacks is required. This guarantees that whenever someone shows a valid signature σ on m, the signer must have created this pair. Note that a randomizable signature cannot be strongly existentially unforgeable against a known message attack, as anyone that has a valid message-signature pair can randomize the signature.

A different notion of unforgeability is F-unforgeability [BCKL08]. Now given some efficiently computable bijection F an attacker must output $F(m), \sigma$ with $\text{Verify}_{pk}(m, \sigma) = 1$, for an m for an m on which he has not previously received a signature. If this is infeasible the scheme is existentially F-unforgeable.

We now describe some frequently used digital signature schemes.

2.7.1 Plain RSA signature

The plain RSA signature [RSA78] is the one of the first instantiations of public key cryptography as proposed in [DH76].

KeyGen Choose large primes p, q and take $n = pq$. Choose e with $1 < e < (p-1)(q-1)$ and $gcd(e,(p-1)(q-1)) = 1$, and calculate $d = \frac{1}{e} \pmod{(p-1)(q-1)}$. The public key is (n, e), the secret key is d. Delete p, q.

Sign(m) To sign $m \in \mathbb{Z}_n$, take $\sigma = m^d \pmod{n}$.

Verify($m;\sigma$) $\sigma^e \stackrel{?}{=} m \pmod{n}$

Plain RSA signatures are existentially forgeable by a key-only attack: take $\sigma \in_R \mathbb{Z}_n$, and let $m = \sigma^e$. Now σ is a valid signature on m.

Plain RSA signatures are homomorphic, which allows selective forgery by a generic chosen message attack. The forger wants to forge a signature on m. He takes m_1, m_2 such that $m_1 \cdot m_2 = m \pmod{n}$, and queries the signing oracle for valid signatures σ_1, σ_2 on m_1, m_2 respectively. Now $\sigma = \sigma_1 \cdot \sigma_2 = m_1^d m_2^d = (m_1 \cdot m_2)^d = m^d \pmod{n}$ is a signature on m.

Both attacks can be mitigated by signing a cryptographic hash of a message instead of the message. Intuitively, this can be seen from the fact that by using the mentioned ways to forge signatures, the attacker must now still find a preimage of the hash function, which is infeasible. Using a secure hash function this signature is generally considered to be secure and standardized in PKCS#1 [JK03], although the security has not been proven. RSA-FDH uses a full domain hash $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_n^*$. This construction is existentially unforgeable against adaptive chosen message attacks under the RSA assumption in the random oracle model, as shown in [BR96, Cor00]. The random oracle model models the output of a hash function as truly random numbers uniform in the range of the hash function (but it is still a function, so hashing the same input multiple times will yield the same output) [BR93]. RSA-PSS uses a probabilistic hash function and can be more tightly reduced to the RSA problem, again using the random oracle model [BR96].

2.7.2 Plain RSA with multiple exponents

In order to mitigate the homomorphic properties of RSA signatures, one can use a fresh e and d for every signature.

KeyGen Choose large primes p, q and take $n = pq$. The public key is n, the secret key is (p, q).

Sign(m) To sign $m \in \mathbb{Z}_n$, take e with $1 < e < (p-1)(q-1)$ and $gcd(e, (p-1)(q-1)) = 1$, compute $d = \frac{1}{e} \pmod{(p-1)(q-1)}$ and set $\sigma = m^d \pmod{n}$. The signature is (σ, e).

Verify($m; \sigma, e$) $\sigma^e \stackrel{?}{=} m \pmod{n}$

The probability that the attacker can get two messages with the same e is negligible, so he cannot abuse the homomorphic properties. In this setup the signer has to take care that the private exponents d are large. As shown in [HGS99] and [SM10], if the attacker can get $e_1, ..., e_k$ with corresponding (but to the attacker unknown) decryption exponents $d_1, ..., d_k$ such that $d_i < n^{\frac{3k-1}{4k+4}}$, the modulus n can be factored. Note that as k grows, d_i must be at least $n^{\frac{3}{4}}$. However, for a large (say 2048 bit) modulus, this is not a problem, since the probability that d_i for a randomly chosen e_i is that small is negligible.

The existential forgery by a key only attack of the plain RSA signature occurs here too, which again can be mitigated by using a hash function.

2.7.3 Weak Boneh-Boyen signature

The weak Boneh-Boyen signature [BB04] is existentially unforgeable against generic chosen message attacks under the q-SDH assumption. Unlike RSA-signatures, it

does not rely on the random oracle model. It is proven to be existentially F-unforgeable against adaptive chosen message attacks with $F(m) = ({g_2}^m, u^m)$, with u some fixed element in $\mathbb{G}_1$ [BCKL08].

KeyGen Let $\mathbb{G}_1 = \langle g_1 \rangle, \mathbb{G}_2 = \langle g_2 \rangle$ of prime order p with bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Take $x \in_R \mathbb{Z}_p^*$. The public key is $pk = (g_1, g_2, v = {g_2}^x)$, the secret key $sk = x$

Sign(m) $\sigma = {g_1}^{1/(x+m)}$

Verify$(m; \sigma)$ $e(\sigma, v \cdot {g_2}^m) \stackrel{?}{=} e(g_1, g_2)$

2.7.4 CL-signature

The CL-signature [CL03] is a randomizable signature scheme secure under the strong RSA assumption, and can sign L messages in a single signature.

KeyGen The signer chooses two large safe primes $p = 2p' + 1, q = 2q' + 1, n = pq$. Let $S \in QR(n)$ be a generator of a subgroup of $QR(n)$ of order $p'q'$. Finally, he takes $Z, R_1, ..., R_L \in_R \langle S \rangle$. The public key is $(n, S, Z, R_1, ..., R_L)$, the secret key is (p, q).

Sign$(m_1, ..., m_L)$ Let the message space be $\{0,1\}^{l_m}$ (i.e. every message $m_i \in \{0,1\}^{l_m}$). Choose a random fresh prime e of bit length l_e such that $l_e \geq l_m + 2$, and a random v. Compute $A = (\frac{Z}{S^v \prod_{i=1}^{L} R_i^{m_i}})^{1/e}$. The signature is (A, e, v).

Verify$(A, e, v; m_1, ..., m_L)$ $A^e \stackrel{?}{=} \frac{Z}{S^v \prod_{i=1}^{L} R_i^{m_i}} \pmod{n}$, and verify the bit lenghts: $m_i \in \{0,1\}^{l_m}$, $e \in \{0,1\}^{l_e}$.

Note that we need that e has a bit length greater than the acceptable interval of m_i, otherwise we can existentially forge signatures using a known message attack. Suppose attacker knows signature (A, e, v) on $m_1, ..., m_L$. He could alter the value of m_i by taking a random r and setting $A' = A \cdot R_i^r$. This would yield a signature on $m_i' = m_i - r \cdot e$. However, when e has a length greater than the allowed range of messages, m_i' can no longer be in the correct range, so verification fails.

Chapter 3

Zero-knowledge Proofs

Zero-knowledge proofs are a concept introduced in [GMR89] and allow one to prove that a statement is true, without revealing anything more than the fact that the statement is true. In practicular, the verifier cannot convince anybody else of the fact that this statement is true. We start this section by explaining the concept using the 'magic cave' example from [QQQ+90]. After this, interactive and non-interactive proof systems are considered individually. Interactive proofs involve multiple rounds of communication between a prover and verifier, whereas a non-interactive proof is a single message from the prover to the verifier.

3.1 Magic Cave example

This example involves a very special and famous cave. After the entrance, it splits into two paths, that lead to different sides of a magic door, as depicted in Figure 3.1. This secret door only opens by saying the correct password. Peggy knows this password, and wants to convince Victor of this. However, she only wants to convince Victor, not anyone else, because she does not want the world to know she can open the famous magic door.

Peggy and Victor proceed as follows. Peggy will enter the cave and walk through path A or B to the magic door, leaving her at the A-side or the B-side of the magic door respectively. After Peggy entered the cave, Victor will walk through the entrance to the intersections (not knowing which way Peggy chose) and flips a coin. He will shout "exit A" in case of heads, and "exit B" in case of tails. Peggy can hear this and will exit the cave through this path. If she is on the A-side of the door and has to exit through A, she can simply walk out. If she's on the A-side but has to exit B, she must pass the secret door, using her password. In the same way, if Peggy chose B and has to exit B, she just walks out, and if she chose B and must exit A, she will go through the door (see Figure 3.2).

Victor will only see Peggy exiting through the correct path, but he does not know whether Peggy went through the magic door or not. However, when doing this repeatedly, Victor will be convinced that Peggy is able to pass the secret door, because otherwise she would fail to exit through the correct path in half of the cases.

Victor is so amazed that he decides to record everything he sees: he flips a coin, and Peggy comes out of the correct path. He shows his recorded video to his friend Trent, in order to convince him that Peggy knows the password to the

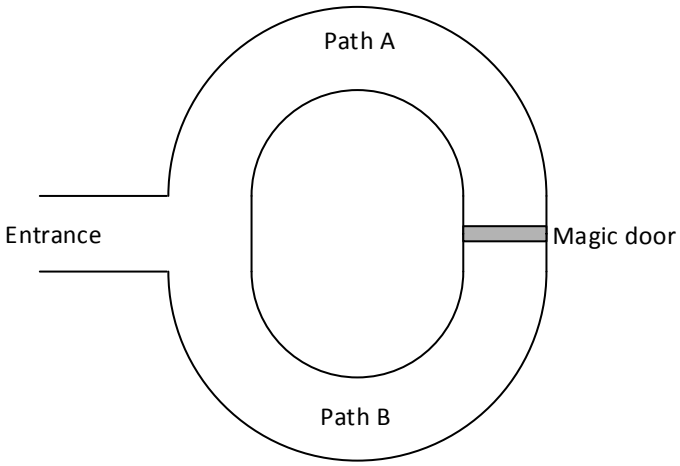


Figure 3.1: The magical cave

secret door. Trent, however, is not convinced: Victor could easily have simulated the whole thing, since anyone that does not know the password will still be able to exit through the correct path in half of the cases. All the failed attempts will simply be cut out of the recording. This is the zero-knowledge aspect of Peggy's proof: everything she shows to Victor could have been simulated by Victor alone, such that a third party could not distinguish Peggy's proof (as recorded in Victor's video) from a simulation (a video involving someone that cannot pass the door, with all the failed attempts cut out). Peggy therefore managed to fully convince Victor that she is able to pass the door, but Victor cannot convince anyone else.

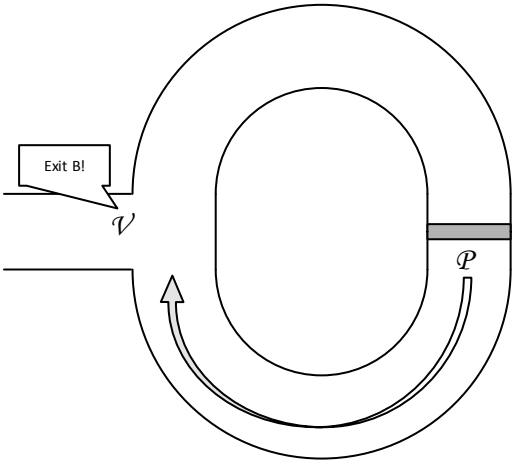
3.2 Interactive Proofs

Given a relation $\mathcal{R} \subseteq V \times W$, and corresponding language $L_{\mathcal{R}} = \{v \in V : \exists_{w \in W}(v, w) \in \mathcal{R}\}$ with $L_{\mathcal{R}} \in NP$, a prover needs to prove to a verifier that some statement v is in this language. Both the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ are probabilistic polynomial time (PPT) algorithms.

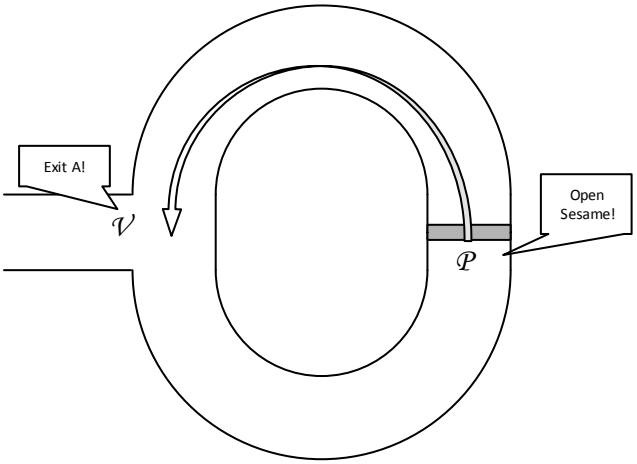
In an interactive proof system, prover $\mathcal{P}$ and verifier $\mathcal{V}$ exchange messages, in order to convince $\mathcal{V}$ that $x \in L_{\mathcal{R}}$. Eventually $\mathcal{V}$ accepts or reject the proof, denoted by $(\mathcal{P}, \mathcal{V})[x] = Accept$ and $(\mathcal{P}, \mathcal{V})[x] = Reject$. Let $view(\mathcal{P}, \mathcal{V})[x]$ denote all the messages exchanged between prover $\mathcal{P}$ and verifier $\mathcal{V}$ in $\mathcal{P}$'s attempt to prove to $\mathcal{V}$ that $x \in L_{\mathcal{R}}$. Since $\mathcal{P}$ and $\mathcal{V}$ are probabilistic algorithms, $view(\mathcal{P}, \mathcal{V})[x]$ is a random variable. From now, we use $\mathcal{P}$ and $\mathcal{V}$ to denote an honest prover and verifier, and $\mathcal{P}^*$ and $\mathcal{V}^*$ to denote a possibly cheating prover and verifier. An honest participant will always stick to the protocol, a cheating participant may deviate from the protocol. We always require:

Completeness A proof from an honest prover will always be accepted by an honest verifier: $\forall x \in L_{\mathcal{R}} : Pr[(\mathcal{P}, \mathcal{V})[x] = Accept] = 1$.

Soundness A cheating prover has a small probability of being accepted by an honest verifier: $\forall x \notin L_{\mathcal{R}} : Pr[(\mathcal{P}^*, \mathcal{V})[x] = Accept] \leq \frac{1}{2}$. By doing repetitions of



(a) Peggy can simply walk out path B



(b) Peggy has to use the magic door

Figure 3.2: Peggy exits the way Victor names

the interactive proof, we can get the success probability of cheating arbitrarily small.

For some relations, we do not only want to prove that some $x \in L_{\mathcal{R}}$, but also that we know some witness w such that $(x, w) \in \mathcal{R}$. Such proofs are called proofs-of-knowledge [BG93]. For such proof systems, the soundness requirement is replaced with the knowledge soundness requirement. Informally, this requirement states that a prover that is able to convince a verifier that he knows a witness, must actually know this witness. This is enforced by the fact that an efficient extractor algorithm that has oracle access to the prover, must be able to compute a witness.

Knowledge Soundness A probabilistic extractor $\mathcal{E}$ with oracle access to a cheating prover (as rewindable turing machine) can compute a witness if the cheating prover has a success probability larger than the knowledge error: let $\epsilon(x) = Pr[(\mathcal{P}^*, \mathcal{V})[x] = Accept]$, we require $\exists_{PPT}\mathcal{E}\forall_{PPT}\mathcal{P}^*\epsilon(x) > \kappa(x) \rightarrow (x, \mathcal{E}_{\mathcal{P}^*}(x)) \in \mathcal{R}$, in at most $\frac{|x|^c}{\epsilon(x)-\kappa(x)}$ steps, for some constant c.

There are multiple notions of zero-knowledge, but the idea always involves simulation. If anyone can simulate conversations between an honest prover and a verifier, such that simulated conversations look just like actual conversations, then a verifier cannot learn anything from a prover, as he can simulate proofs on his own. The formal definitions are:

Perfect zero-knowledge Simulated conversations and actual conversations are equally distributed:
$\forall_{PPT}\mathcal{V}^*\exists_{PPT}S : \forall x \in L : \Delta(view(\mathcal{P}, \mathcal{V}^*)[x], S[x]) = 0$

Statistical zero-knowledge Simulated conversations and actual conversations are statistically indistinguishable:
$\forall_{PPT}\mathcal{V}^*\exists_{PPT}S : \forall x \in L : \Delta(view(\mathcal{P}, \mathcal{V}^*)[x], S[x]) \cong 0$

Computational zero-knowledge Simulated conversations and actual conversations are computationally indistinguishable, i.e. all PPT binary distinguishers $\mathcal{D}$ have a negligible advantage distinguishing simulated from actual conversations:
$\forall_{PPT}\mathcal{V}^*\exists_{PPT}S : \forall_{PPT}\mathcal{D} : \forall x \in L : Adv_D(view(\mathcal{P}, \mathcal{V}^*)[x], S[x]) \cong 0$

Perfect Honest verifier zero-knowledge Simulated conversations and actual conversations between an honest prover and verifier are equally distributed: $\exists_{PPT}S :$
$\forall x \in L : \Delta(view(\mathcal{P}, \mathcal{V})[x], S[x]) = 0$

Statistical Honest verifier zero-knowledge Simulated conversations and actual conversations between an honest prover and verifier are statistically indistinguishable: $\exists_{PPT}S : \forall x \in L : \Delta(view(\mathcal{P}, \mathcal{V})[x], S[x]) \cong 0$

Computational Honest verifier zero-knowledge Simulated conversations and actual conversations between an honest prover and verifier are computationally indistinguishable, i.e. all PPT binary distinguishers $\mathcal{D}$ have a negligible advantage distinguishing simulated from actual conversations:
$\exists_{PPT}S : \forall_{PPT}\mathcal{D} : \forall x \in L : Adv_D(view(\mathcal{P}, \mathcal{V})[x], S[x]) \cong 0$

A weaker notion than zero-knowledge is witness indistinguishability. This states that all valid witnesses for a statement will result in indistinguishable conversations. Note that proofs for languages with a single witness for a statement are

Prover $h = g^x$	Public $g, \mathbb{G}$	**Verifier** h
$u \in_R \mathbb{Z}_q$ $t = g^u$	$\xrightarrow{t}$	
	$\xleftarrow{c}$	$c \in_R \{0,1\}$
$s = u + cx \pmod q$	$\xrightarrow{s}$	
		$g^s \stackrel{?}{=} th^c$

Figure 3.3: Schnorr's identification protocol

always witness-indistinguishable, because all witnesses are equally likely. A zero-knowledge proof is always witness indistinguishable, because if you learn nothing from the proof, you cannot learn anything about the witness.

3.2.1 Schnorr's identification protocol

Schnorr's identification protocol [Sch91] is an interactive proof system for language $L_{\mathcal{R}}$ with $\mathcal{R} = \{(h, x) : h = g^x\}$, given a multiplicative group G of order q with generator g. The protocol is shown in Figure 3.3.

Completeness holds trivially: $g^{u+cx} = g^u g^{cx} = th^c$.

Schnorr's identification protocol is a proof-of-knowledge with knowledge error $\kappa(x) = \frac{1}{2}$. We show that if an attacker can answer two distinct challenges on the same t-value, we can extract a witness. [Dam02] shows that special soundness implies proof-of-knowledge with knowledge error $2^{-|c|}$, with $|c|$ the challenge size. Given two accepting conversations $(t; c_1; s_1)$ and $(t; c_2, s_2)$ with $c_1 \neq c_2$, we can extract a witness. W.l.o.g. $c_1 = 0$ and $c_2 = 1$. Because these are accepting conversations, we have $g^{s_1} = t$ and $g^{s_2} = th$, so $g^{s_2 - s_1} = h$, and $log_g(h) = s_2 - s_1 = x \pmod q$.

This protocol is perfect zero-knowledge. The simulator first takes a random $c \in_R \{0,1\}$. If $c = 0$, the simulator sets $t = g^u$, and $s = u$ would be a correct answer for the challenge. If $c = 1$, the simulator sets $t = g^u/h$, and $s = u$. The simulator sends t to $\mathcal{V}^*$. $\mathcal{V}^*$ sends challenge c' in return. If $c = c'$, the simulator has simulated an accepting conversation. If $c \neq c'$, the simulator rewinds $\mathcal{V}^*$ and starts over by making a new guess for c. Since $c = c'$ in half of the cases, on average it will take 2 attempts to simulate a conversation.

Since a prover that does not know $log_g(h)$ can pass half the challenges, this protocol needs to be repeated k times, such that a cheating verifier has probability 2^{-k} to pass them all. This is unpractical. To improve efficiency, one can take $c \in_R \mathbb{Z}_q$ instead of $c \in_R \{0,1\}$. A cheating prover can still prepare $t = g^u/h^{-c}$ for one challenge, but now the probability of guessing c correctly is $1/q$: the knowledge error is now only $1/q$. However, this protocol is no longer perfect zero-knowledge. To simulate conversations, the simulator must guess c correctly, which is now infeasible in polynomial time. If we assume the verifier to be honest, the verifier will take $c \in_R \mathbb{Z}_q$. Conversations between a prover and an honest-verifier have distribution $\{(t; c; s) : u, c \in_R \mathbb{Z}_q, t = g^u, s = u + cx \pmod q\}$. These can be simulated: $\{(t; c; s) : s, c \in_R \mathbb{Z}_q, t = g^s h^{-c}\}$ is the same distribution. Because this protocol is perfect honest verifier zero-knowledge, this protocol is referred to as Schnorr's honest verifier zero-knowledge protocol.

3.2.2 Okamoto's identification protocol

Okamoto's identification protocol [Oka93] is a variantion on Schnorr's honest-verifier zero knowledge protocol. It is an interactive proof system for proving discrete logarithm representations: $L_{\mathcal{R}}$ with $\mathcal{R} = \{(h; x_1, x_2) : h = g_1^{x_1} g_2^{x_2}\}$, given a multiplicative group G of order q with generator g_1 and g_2 (this protocol can actually prove discrete log representations for a list of L generators, for simplicity we show it with two generators). The protocol is shown in Figure 3.4.

Completeness: $g_1^{u_1+cx_1} g_2^{u_2+cx_1} = g_1^{u_1} g_1^{cx_1} g_2^{u_2} g_2^{cx_2} = th^c$.

Witness extraction works the same as in Schnorr's HVZK protocol: if a prover can answer two distinct challenges, he must know some x_1, x_2 with $h = g_1^{x_1} g_2^{x_2}$. Suppose a cheating prover could answer both challenges c, c' with $c \neq c'$ for some t, the prover must know s_1, s_2, s_1', s_2' with $g_1^{s_1} g_2^{s_2} = th^c$ and $g_1^{s_1'} g_2^{s_2'} = th^{c'}$. This means the prover knows $g_1^{s_1-s_1'} g_2^{s_2-s_2'} = h^{c-c'}$, so he knows representation $x_1 = \frac{s_1-s_1'}{c-c'} \pmod q$, $x_2 = \frac{s_2-s_2'}{c-c'} \pmod q$. Again by [Dam02], this is a proof-of-knowledge with knowledge error $\kappa = 1/q$.

Okamoto's identification protocol is perfect HVZK: we can simulate conversations $\{(t; c; s_1, s_2) : s_1, s_2, c \in_R \mathbb{Z}_q, t = g_1^{s_1} g_2^{s_2} h^{-c}\}$, which is equal to actual conversations $\{(t; c; s_1, s_2) : u_1, u_2, c \in_R \mathbb{Z}_q, t = g_1^{u_1} g_2^{u_2}, s_1 = u_1 + cx_1 \pmod q, s_2 = u_2 + cx_2 \pmod q\}$, as every conversations occurs with probability q^{-3}.

Prover $h = g_1^{x_1} g_2^{x_2}$	Public $g_1, g_2, \mathbb{G}$	**Verifier** h
$u_1, u_2 \in_R \mathbb{Z}_q$		
$t = g_1^{u_1} g_2^{u_2}$	$\xrightarrow{t}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_q$
$s_1 = u_1 + cx_1 \pmod q$		
$s_2 = u_2 + cx_2 \pmod q$	$\xrightarrow{s_1, s_2}$	
		$g_1^{s_1} g_2^{s_2} \stackrel{?}{=} th^c$

Figure 3.4: Okamoto's identification protocol

3.2.3 Sigma protocol composition

Sigma protocols [CDS94, Cra97] are a generalization of Schnorr's honest verifier zero-knowledge protocol. These protocols can be composed in multiple ways, which yields sigma protocols for more complex relations. A sigma protocol has three steps: the prover sends the t-values to the verifier, the verifier responds with a challenge c, and the prover answers the challenge in the s-values, and must fulfill three requirements: completeness, honest-verifier zero-knowledge, and special soundness. Special soundness requires that if an attacker can answer two distinct challenges for a single t-value, we can extract a witness. As shown before, this also guarantees that it is a proof-of-knowledge with knowledge error 2^{-c}, with c the challenge size. Sigma protocols can easily be created for relations about discrete logarithms in multiplicative groups. Schnorr's honest-verifier zero knowledge protocol and Okamoto's protocol are instances of sigma protocols.

Prover $h_1 = {g_1}^{x_1}, h_2 = {g_2}^{x_2}$	Public $g_1, g_2, \mathbb{G}$	**Verifier** h_1, h_2
$u_1, u_2 \in_R \mathbb{Z}_q$ $t_1 = {g_1}^{u_1}$ $t_2 = {g_2}^{u_2}$	$\xrightarrow{t_1, t_2}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_q$
$s_1 = u_1 + cx_1 \pmod q$ $s_2 = u_2 + cx_2 \pmod q$	$\xrightarrow{s_1, s_2}$	
		${g_1}^{s_1} \stackrel{?}{=} t_1 {h_1}^c$ ${g_2}^{s_2} \stackrel{?}{=} t_2 {h_2}^c$

Figure 3.5: And-composition of sigma protocols

And-composition

Given sigma protocols for relations $\mathcal{R}_1$ and $\mathcal{R}_1$, we can create a sigma protocol for relation $\mathcal{R} = \{(v_1, v_2, w_1, w_2) \mid (v_1, w_1) \in \mathcal{R}_1 \wedge (v_2, w_2) \in \mathcal{R}_2\}$. Let the t-values be the t-values of the protocols for $\mathcal{R}_1$ and $\mathcal{R}_2$ combined. The verifier sends a single challenge, and the s-values are the s-values from the protocols of $\mathcal{R}_1$ and $\mathcal{R}_2$ combined. The verifier checks both relations individually, and accepts when both hold. Completeness holds, because completeness holds for the sub protocols used. Soundness holds, because we can extract a witness for both relations (using the witness extraction of the sub protocols) given two accepting conversations on a single t-value. We can simulate by combining the simulators of the sub protocols.

An example is shown in Figure 3.5.

Equality-composition

Given sigma protocols for relations $\mathcal{R}_1$ and $\mathcal{R}_1$, we can create a sigma protocol that proves knowledge of a single witness that satisfies both relations:
$\mathcal{R} = \{(v_1, v_2, w) \mid (v_1, w) \in \mathcal{R}_1 \wedge (v_2, w) \in \mathcal{R}_2\}$. This works very similarly to the and-composition, but now we use the same randomness for both t-values, and the prover can therefore send a single s-value such that both relations are satisfied. This single s-value allows us to extract one witness that satisfies both relations. Similar to and-composition, completeness and simulation follows form the completeness and simulation of the sub protocols.

An example is provided in Figure 3.6. This composition was introduced in [CP93].

Or-composition

We can also create sigma protocols for a relation that is a disjunction of two relations for which we have sigma protocols: $\mathcal{R} = \{(v_1, v_2, w_1, w_2) \mid (v_1, w_1) \in \mathcal{R}_1 \vee (v_2, w_2) \in \mathcal{R}_2\}$. The prover only needs to know a witness for one of the relations, but the proof must hide which relation that is. In order to do so, the prover must prove both

Prover $h_1 = {g_1}^x, h_2 = {g_2}^x$	Public $g_1, g_2, \mathbb{G}$	**Verifier** h_1, h_2
$u \in_R \mathbb{Z}_q$ $t_1 = {g_1}^u$ $t_2 = {g_2}^u$	$\xrightarrow{t_1, t_2}$	
		$c \in_R \mathbb{Z}_q$
	$\xleftarrow{c}$	
$s = u + cx \pmod q$	$\xrightarrow{s}$	
		${g_1}^s \stackrel{?}{=} t_1 {h_1}^c$ ${g_2}^s \stackrel{?}{=} t_2 {h_2}^c$

Figure 3.6: Equality-composition of sigma protocols

Prover $h_1 = {g_1}^{x_1}$	Public $g_1, g_2, \mathbb{G}$	**Verifier** h_1, h_2
$c_2, u_1, s_2 \in_R \mathbb{Z}_q$ $t_1 = {g_1}^{u_1}$ $t_2 = {g_2}^{s_2} h^{-c_2}$	$\xrightarrow{t_1, t_2}$	
		$c \in_R \mathbb{Z}_q$
	$\xleftarrow{c}$	
$c_1 = c - c_2 \pmod q$ $s_1 = u_1 + c_1 x_1 \pmod q$	$\xrightarrow{c_1, c_2, s_1, s_2}$	
		$c_1 + c_2 \stackrel{?}{=} c \pmod q$ ${g_1}^{s_1} \stackrel{?}{=} t_1 {h_1}^{c_1}$ ${g_2}^{s_2} \stackrel{?}{=} t_2 {h_2}^{c_2}$

Figure 3.7: Or-composition of sigma protocols

statements as in and-composition, but is allowed to 'cheat' in one of them. The prover receives a challenge c, and then picks c_1 and c_2, with $c_1 + c_2 = c \pmod q$, and uses challenge c_1 for proving $\mathcal{R}_1$ and c_2 for $\mathcal{R}_2$. Suppose the prover knows a witness for $\mathcal{R}_1$, he must simulate a proof for $\mathcal{R}_2$. He can do this by choosing c_2 before receiving c, and use the simulation of the sub protocol for $\mathcal{R}_2$ to simulate this part of the proof. He then receives c, and sets $c_1 = c - c_2 \pmod q$. Along with the s-values, he sends c_1 and c_2, and the verifier will check $c_1 + c_2 \stackrel{?}{=} c$. The verifier cannot distinguish whether the prover fixed c_1 or c_2 first. An or-composition for relation $\mathcal{R} = \{(v_1, v_2, w_1, w_2) \mid v_1 = {g_1}^{w_1} \vee v_2 = {g_2}^{w_2}\}$ is shown in Figure 3.7. The prover knows witness (h, x) for $\mathcal{R}_1$, and simulates a proof for $\mathcal{R}_2$.

Extraction now works slightly different. Given two accepting conversations $(t_1, t_2; c; c_1, c_2, s_1, s_2)$ and $(t_1, t_2; c'; c_1', c_2', s_1', s_2')$ with $c \neq c'$, we must have $c_1 \neq c_1'$ or $c_2 \neq c_2'$. In the first case, we use the extractor of the first sub protocol to extract a witness for $\mathcal{R}_1$. In the second case, we use the extractor of the second sub pro-

tocol and get a witness for $\mathcal{R}_2$. Either witness is sufficient for $\mathcal{R}$ because this is a disjunction.

We will use the notation introduced in [CS97] to denote sigma proofs-of-knowledge proving knowledge of witness $\chi_1, ...\chi_n$ with $(v_1, ..., v_m; \chi_1, ..., \chi_n) \in \mathcal{R}$: $PK\{(\chi_1, ..., \chi_n) : (v_1, ..., v_m; \chi_1, ..., \chi_n) \in \mathcal{R}\}$, although in some cases we will not use Greek letters for all the values we prove knowledge of.

3.2.4 Discrete logarithms in groups of unknown order and interval proofs

So far we only considered proving statements about discrete logarithms in groups of known order. This allows us to take t uniformly at random in the group, and the s-value modulo the order of the group. However, when using QR_n groups with secret factorization $n = pq = (2p' + 1)(2q' + 1)$, this is not possible. The order of the group generated by g is $p'q'$, but this cannot be published as this would factor n. We will show how $PK\{(\chi) : g^\chi = h \pmod n\}$ changes when it takes place in a composite order group. Let l_n denote the bit length of the modulus n, l_c the length of the challenge. Constant l_ϕ is used to gain statistical zero-knowledge.

1. The prover chooses $u \in_R \{0,1\}^{l_n+l_c+l_\phi}$, and sends $t = g^u \pmod n$ to the verifier
2. The verifier chooses challenge $c \in_R \{0,1\}^{l_c}$
3. The prover computes $s = u - cx$ (in $\mathbb{Z}$)
4. The verifier checks $t \stackrel{?}{=} g^s h^c \pmod n$

We cannot take $u \in_R QR_n$, as we do not know the order of the group. However, we still need to take t and s such that they do not reveal information about the witness. By taking l_ϕ sufficiently large, the s-value is statistically close to uniform random in $\{0,1\}^{l_n+l_c+l_\phi}$. This way we can simulate proofs that are statistically close to real proofs, making this statistically honest-verifier zero-knowledge.

This is a proof-of-knowledge for $l_c = 1$. Given $(t; c; s), (t; c'; s')$ accepting conversations, w.l.o.g. we have $c = 0$ and $c' = 1$, then the prover knows $log_g(h) = s - s' = x$. Because $l_c = 1$, we need many repetitions, which is impractical. For $l_c > 1$, formally this is not a proof-of-knowledge. Again, we have $(t; c; s), (t; c'; s')$ accepting conversations, $c \neq c'$. Assume (w.l.o.g.) that $c' > c$, then $log_g(h) = \frac{s-s'}{c'-c} \pmod{p'q'}$, but the extractor cannot compute this since it does not know $p'q'$. Under the Strong RSA assumption a prover that does not know $p'q'$ can only create proofs with $c' - c$ a divisor of $s - s'$, as proven in [CS02]. Let $u(c' - c) = (s - s')$, we have $g^{u(c'-c)} = h^{c'-c} = g^{s-s'}$, so we have extracted $log_g(h) = u$. A prover with knowledge of the factorization of n can create s, s' such that it is not divided by $c' - c$, and consequently the extractor cannot extract a witness. Such a prover exists, because there is a PPT algorithm which has the factorization of n encoded into the machine. However, since this only concerns provers that know the factorization of n, which we assume to be secret, we still call this a proof-of-knowledge.

In groups of unknown order, we can enhance proofs such that we also prove things about the bit length of the witness, as introduced in [CFT98]. To prove that $x \in \pm\{0,1\}^{l_x}$, the verifier also checks $s \stackrel{?}{\in} \pm\{0,1\}^{l_x+l_c+l_\phi+1}$ in the final step of the

proof. Only a prover that knows a sufficiently small witness can create such proofs with a high probability. Again this is formally not a proof-of-knowledge, but we denote this by $PK\{(\chi) : g^{\chi} = h \wedge \chi \in \pm\{0,1\}^{l_x}\}$.

Using this, we can prove that the witness lies in some interval. To prove that the witness must lie in range $[a, b]$, we first move the witness to the middle of the interval by changing the statement we prove. For example, instead of proving $PK\{(\chi) : g^{\chi} = h\}$, we prove the equivalent $PK\{(\chi) : g^{\chi} = \frac{h}{g^{(a+b)/2}}\}$. If the prover knows $g^x = h$, then he also knows $g^{x-\frac{a+b}{2}} = \frac{h}{g^{(a+b)/2}}$. If the witness was in the range $[a, b]$ for the original statement, it will be in the range $[-\frac{b-a}{2}, \frac{b-a}{2}]$. Suppose $\frac{b-a}{2}$ has bit length l_r, we can prove that the witness lies in interval $[a, b]$ by proving $PK\{(\chi) : g^{\chi} = \frac{h}{g^{(a+b)/2}} \wedge \chi \in \pm\{0,1\}^{l_r}\}$. This proof will be denoted by $PK\{(\chi) : g^{\chi} = h \wedge \chi \in [a, b]\}$.

3.2.5 Secret modular arithmetic

In [CM99] a way to do modular arithmetic in proofs of knowledge is described. Again we use a composite order group, and a group for Pedersen commitments.

To prove knowledge of a, b, d, n with $a + b = d \pmod{n}$ without revealing any of the values, the prover first creates Pedersen commitments to a, b, d, n giving $c_a = g^a h^{u_a}, c_b = g^b h^{u_b}, c_d = g^d h^{u_d}, c_n = g^n h^{u_n}$, and sends these to the verifier. The prover proves knowledge of all openings and that a, b, d, n have the correct size. Finally he shows that $d - (a + b) = kn$ for some k, by proving knowledge of k, λ such that $\frac{c_d}{c_a c_b} = {c_n}^k h^{\lambda}$. More precise, let $a, b, c, d \in \pm\{0,1\}^l$, the prover proves

$$
\begin{aligned}
&PK\{(a, b, d, n, u_a, u_b, u_d, u_n, k, \lambda) : \\
&\quad c_a = g^a h^{u_a} \wedge a \in \pm\{0,1\}^l && \wedge \\
&\quad c_b = g^b h^{u_b} \wedge b \in \pm\{0,1\}^l && \wedge \\
&\quad c_d = g^d h^{u_d} \wedge d \in \pm\{0,1\}^l && \wedge \\
&\quad c_n = g^n h^{u_n} \wedge n \in \pm\{0,1\}^l && \wedge \\
&\frac{c_d}{c_a c_b} = {c_n}^k h^{\lambda} \wedge k \in \pm\{0,1\}^l \\
&\quad \}
\end{aligned}
$$

This proof will be denoted by $PK\{(a, b, d, n) : a + b = d \pmod{n}\}$.

In a similar way, we can prove that $a \cdot b = d \pmod{n}$. The prover knows $a \cdot b = d + k \cdot n$, so $(g^a)^b = g^d \cdot (g^n)^k$. We now prove

$$
\begin{aligned}
&PK\{(a, b, d, n, u_a, u_b, u_d, u_n, k, \lambda) : \\
&\quad c_a = g^a h^{u_a} \wedge a \in \pm\{0,1\}^l && \wedge \\
&\quad c_b = g^b h^{u_b} \wedge b \in \pm\{0,1\}^l && \wedge \\
&\quad c_d = g^d h^{u_d} \wedge d \in \pm\{0,1\}^l && \wedge \\
&\quad c_n = g^n h^{u_n} \wedge n \in \pm\{0,1\}^l && \wedge \\
&\quad c_d = {c_b}^a {c_n}^k h^{\lambda} \wedge k \in \pm\{0,1\}^l \\
&\quad \}
\end{aligned}
$$

We will refer to this proof by $PK\{(a,b,d,n) : a \cdot b = d \pmod n\}$

Finally, we can create proofs-of-knowledge of $a^b = d \pmod n$. Let b_i denote the binary representation of b, $b = \sum_{i=0}^{l-1}(b_i \cdot 2^i)$, so we have $a^b \pmod n = \prod_{i=0}^{l-1}(b_i \cdot a^{2^i}) \pmod n$.

The prover starts by calculating $\{\lambda_i = a^{2^i} \pmod n\}_{0<i\leq l-1}$, and proves knowledge of this using the secret modular multiplication:

$$\begin{aligned} PK\{&(a, \lambda_1 \ldots \lambda_l, n) : \\ &a \cdot a = \lambda_1 \pmod n && \wedge \\ &\lambda_1 \cdot \lambda_1 = \lambda_2 \pmod n && \wedge \\ &\ldots && \wedge \\ &\lambda_{l-2} \cdot \lambda_{l-2} = \lambda_{l-1} \pmod n \\ \} & \end{aligned}$$

Next, the prover proves that bits b_i indeed form b, by proving $PK\{(b_1 \ldots b_{l-1}, b) : b = \sum_{i=0}^{l-1}(b_i \cdot 2^i)\}$ using secret modular multiplication.

Now the prover will prove knowledge of $a^b \pmod n = \prod_{i=0}^{l-1}(b_i \cdot a^{2^i}) \pmod n$ bit by bit: Let $\pi_i = \prod_{j=1}^{i}(b_j \cdot a^{2^j}) \pmod n$. The first step is proving $(b_0 = 0) \wedge (\pi_0 = 1) \vee (b_0 = 1) \wedge (\pi_0 = a)$. For every next bit of b we show $(b_i = 0) \wedge (\pi_i = \pi_{i-1}) \vee (b_i = 1) \wedge (\pi_i = \pi_{i-1} \cdot \lambda_i \pmod n)$. Because $\pi_{l-1} = a^b \pmod n = d$, the last step of the proof is $(b_{l-1} = 0) \wedge (d = \pi_{l-2}) \vee (b_{l-1} = 1) \wedge (d = \pi_{l-2} \cdot \lambda_{l-1} \pmod n)$.

All these steps combined will be denoted by $PK\{(a,b,d,n) : a^b = d \pmod n\}$. Such a proof requires about $7l$ modular exponentiations.

Secret modular exponentiation with public exponent

[CM99] only considers proofs where all values are kept secret. In some scenarios only part of these values is required to remain secret. We now show how the secret modular exponentiation changes and becomes much more efficient when the exponent is public.

We can create proofs of $a^b = d \pmod n$ where we keep a, d, n secret, and b is a public constant of bit length l. Because b is public, we no longer have to commit to the bits of b and prove that the bits together form b. The or-composition based on whether bit b_i is 0 or 1 is no longer needed, again because the bits are public. This yields an efficient protocol. This allows us to prove knowledge of an RSA signature $\sigma^3 = m \pmod n$, while keeping all values secret, using only two secret modular multiplications:

$$\begin{aligned} &PK\{(\sigma, m, n, \alpha) : \\ &\sigma \cdot \sigma = \alpha \pmod n && \wedge \\ &\sigma \cdot \alpha = m \pmod n \\ &\} \end{aligned}$$

Let us write out the full proof:

$$
\begin{aligned}
&PK\{(\sigma, m, n, \alpha, u_\sigma, u_m, u_n, u_\alpha, k, k'\lambda, \lambda') : \\
&\quad c_\sigma = g^\sigma h^{u_\sigma} \wedge \sigma \in \pm\{0,1\}^l && \wedge \\
&\quad c_m = g^m h^{u_m} \wedge m \in \pm\{0,1\}^l && \wedge \\
&\quad c_n = g^n h^{u_n} \wedge n \in \pm\{0,1\}^l && \wedge \\
&\quad c_\alpha = g^\alpha h^{u_\alpha} \wedge \alpha \in \pm\{0,1\}^l && \wedge \\
&\quad c_\alpha = {c_\sigma}^\sigma {c_n}^k h^\lambda \wedge k \in \pm\{0,1\}^l \\
&\quad c_m = {c_\alpha}^\sigma {c_n}^{k'} h^{\lambda'} \wedge k' \in \pm\{0,1\}^l \\
&\quad \}
\end{aligned}
$$

We will later see that this proof requires the prover to take only 16 modular exponentiations.

3.3 Non-interactive proofs

For non-interactive zero-knowledge (NIZK) proofs, we want the same three properties that we require for interactive proofs: completeness, soundness and zero-knowledge. However, we cannot have all three trivially, as shown in [GO94]. Suppose we have such a system for language $L_\mathcal{R}$ and we have probabilistic polynomial time algorithms Prove, Verify, and Simulate, then $L_\mathcal{R}$ is a trivial language. We can decide whether $x \in L_\mathcal{R}$ by simulating a proof for x and running the Verify algorithm on it. If $x \in L_\mathcal{R}$, by the fact that a simulated proof is indistinguishable from a real proof and by completeness, the Verify will accept. If $x \notin L_\mathcal{R}$, by soundness Verify will not accept x. Because these three algorithms are polynomial time, anyone can decide whether x is in the language in polynomial time.

To construct NIZK for more than trivial languages, the notion of a common-reference string (*crs*) has been introduced by Blum et al. [BFM88]. In this setting, the prover and verifier must fist have some shared random bits, the *crs*. Later, the prover can create a NIZK proof for some statement using this *crs*, which will convince the verifier. This *crs* can either be created using some secure multiparty computation, or generated by a trusted third party, as long as the *crs* is truly random and cannot be influenced by either the prover of the verifier. In other constructions the *crs* may describe a group or group elements.

The Prove and Verification algorithms now include an argument *crs*. When the prover uses the correct random *crs*, he cannot simulate so we have soundness. To simulate proofs (and achieve zero-knowledge), a simulation *crs* is generated, which also gives a trapdoor τ. Using this *crs* and τ, a verifier can simulate a proof. When the simulated *crs* is (computationally) indistinguishable from a real *crs*, simulated proofs are indistinguishable from real proofs, so we have simulation.

3.3.1 Fiat-Shamir Heuristic

For some applications we prefer non-interactive proofs over interactive proofs. The Fiat-Shamir heuristic [FS87] can be used to transform any sigma-protocol into a non-interactive proof. Let $\mathcal{H}$ be a secure hash function that maps to the challenge space. Instead of asking the verifier for a challenge in the second step, the challenge

is computed as the hash of the t-values. For the non interactive variant of Schnorr's HVZK protocol the challenge would be $c = \mathcal{H}(t)$. The resulting proof is $(c; t)$ which is verified by $c \stackrel{?}{=} \mathcal{H}(g^s h^{-c})$.

We can use non-interactive sigma proofs as a digital signature by adding a message m to the input of the hash function: $c = \mathcal{H}(t, m)$. Verification is done by $c \stackrel{?}{=} \mathcal{H}(g^s h^{-c}; m)$. This is called a signature proof-of-knowledge, and written abstractly as $SPK\{(\chi) : h = g^\chi\}(m)$, again using the notation from [CS97].

If $\mathcal{H}$ is modeled as a random oracle, this is equivalent to the interactive version, because the prover must first choose t-values, and then receive a challenge uniformly random in the challenge space. A security proof in the random oracle model [BR93] was provided in [PS96].

This is zero-knowledge when we let the *crs* describe the random oracle $\mathcal{H}$. A simulator can fix the challenge c beforehand and can prepare his t-value accordingly (using the simulation of the interactive version of the proof).

3.3.2 Groth-Sahai proofs

Groth and Sahai [GS08] constructed a way to create NIZK proofs of statements about bilinear group elements. These statements could already be NIZK proved, since there are constructions that allow NIZK proofs of NP-complete languages such as circuit satisfiability (e.g. [GOS06, GOS12]). Any NP statement can then be proven using a NP reduction to circuit satisfiability. Although this reduction is polynomial, it is still highly inefficient.

GS-proofs can prove many different statements in bilinear groups under different assumptions. We will first focus on proofs about pairing product equations under the subgroup decision assumption, after which we describe the construction in a more generic way. A pairing product equation in multiplicative groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ and pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ has the following form:

$$\prod_{i=1}^{n} e(a_i, y_i) \cdot \prod_{i=1}^{m} e(x_i, b_i) \cdot \prod_{i=1}^{m} \prod_{j=1}^{n} e(x_i, y_i)^{\gamma_{i,j}} = t_T$$

where constants $a_i \in \mathbb{G}_1, b_i \in \mathbb{G}_2, \gamma_{i,j} \in \mathbb{Z}_n, t_T \in \mathbb{G}_T$ define the pairing product equation, and $x_i \in \mathbb{G}_1, y_i \in \mathbb{G}_2$ are variables.

Let us consider a small example. The *crs* describes a composite order group $\langle g \rangle = \mathbb{G}$ of order $n = pq$ with p, q secret primes, G_T, bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. In such a group, some elements will generate $\mathbb{G}$, some will generate the order q subgroup $\mathbb{G}_q$ and some will generate the p order subgroup $\mathbb{G}_p$. The *crs* also describes an element u which generates the order q subgroup $\mathbb{G}_q$. Suppose a prover wants to prove knowledge of x, y such that the simple pairing product equation $e(a, y)e(x, y) = t_T$ holds. The prover first commits to x and y by creating $c_x = xu^r, c_y = yu^s$ with $r, s \in_R \mathbb{Z}_n$. These commitments are perfectly binding in subgroup $\mathbb{G}_p$, because u only has effect on $\mathbb{G}_q$. To prove that the values inside commitments c_x and c_y satisfy the equation, the prover creates a proof $\pi = a^s x^s y^r u^{rs}$, which is

verified by $e(a,c_y)e(c_x,c_y) \stackrel{?}{=} t_T e(u,\pi)$. Completeness holds:

$$\begin{aligned}
e(a,y)e(x,y) &= t_T \\
e(a,yu^s)e(xu^r,yu^s) &= t_T e(a,u^s)e(u^r,y)e(xu^r,u^s) \\
e(a,yu^s)e(xu^r,yu^s) &= t_T e(u,a^s)e(u,y^r)e(u,x^s u^{rs}) \\
e(a,c_y)e(c_x,c_y) &= t_T e(u,a^s x^s y^r u^{rs}) \\
e(a,c_y)e(c_x,c_y) &= t_T e(u,\pi)
\end{aligned}$$

This solution is sound: let $\lambda = \lambda^2 \pmod n$ with $\lambda = 1 \pmod p, \lambda = 0 \pmod q$. By raising both sides to λ^2, all powers of u disappear and we have soundness in $\mathbb{G}_p$:

$$\begin{aligned}
e(a,c_y)e(c_x,c_y) &= t_T e(u,\pi) \\
e(a,c_y)^{\lambda^2}e(c_x,c_y)^{\lambda^2} &= {t_T}^{\lambda^2} e(u,\pi)^{\lambda^2} \\
e(a^\lambda,{c_y}^\lambda)e({c_x}^\lambda,{c_y}^\lambda) &= {t_T}^\lambda e(u^\lambda,\pi^\lambda) \\
e(a^\lambda,y^\lambda)e(x^\lambda,y^\lambda) &= {t_T}^\lambda
\end{aligned}$$

Note that the secret factorization of n is required to efficiently compute λ. This means that we have soundness, but a verifier cannot extract the witness.

We now show that this proof is zero-knowledge when $t_T = 1$. For zero-knowledge we use a simulation *crs*. This *crs* only differs in the order of $\langle u \rangle$, which now has order n instead of q. A simulation trapdoor τ with $g = u^\tau$ is also generated. By the subgroup decision assumption this is indistinguishable from a real *crs*. In this setting, commitments are perfectly hiding. They are not binding, since using τ one can come up with many openings for a single commitment: $Commit(x, open) = Commit(x^{-\delta\tau}, open + \delta)$. For some commitments c_x, c_y, there is one unique π that satisfies $e(a,c_y)e(c_x,c_y) = t_T e(u,\pi)$. This shows that it is independent of the openings of c_x, c_y, all the witnesses used map to the same proof. Therefore, the proof is witness-indistinguishable. However, we still need one witness to simulate a proof. For $t_T = 1 = e(g,g)^0$, we know such a witness: $x_i = y_i = 1$, so we can generate a valid proof. Since we can do so for every commitment, and the proof is valid for every valid opening of the commitments, we can simulate proofs and we have zero-knowledge.

We now describe the GS proof in a more generic way, which allows multiple instantiations under different assumptions. We have some ring R and R-modules $A_1, A_2, A_T, B_1, B_2, B_T, C_1, C_2, C_T$ (using additive notation) with bilinear maps $f_A : A_1 \times A_2 \to A_T$, $f_B : B_1 \times B_2 \to B_T$, $f_C : C_1 \times C_2 \to C_T$. We also have an efficiently computable linear maps $\iota_1, \iota_2, \iota_T$ that map elements from A_1, A_2, A_T into B_1, B_2, B_T respectively, and not efficiently computable linear maps p_1, p_2, p_T that map elements from B_1, B_2, B_T into C_1, C_2, C_T respectively. Figure 3.8 shows this abstract setup and how the concrete example for pairing product equations using the SD assumption instantiates the abstract setup.

The witnesses are in A_1 and A_2. The proof will take place in the B-modues. Soundness will be in the C-modules. The *crs* describes $u_1, ..., u_I \in B_1$, $v_1, ..., v_J \in B_2$. A commitment to a value $x \in A_1$ is $\iota_1(x) \cdot \sum_{i=1}^{I} r_i u_i$, with $r_1, ..., r_I \in_R R$. Similarly, a commitment to $y \in A_2$ is $\iota_2(x) \cdot \sum_{j=1}^{J} r_j v_j$, with $r_1, ..., r_J \in_R R$. The commitments

$$\begin{array}{ccccc} A_1 & \times & A_2 & \xrightarrow{f_A} & A_T \\ \iota_1 \downarrow & & \iota_2 \downarrow & & \iota_T \downarrow \\ B_1 & \times & B_2 & \xrightarrow{f_B} & B_T \\ p_1 \downarrow & & p_2 \downarrow & & p_T \downarrow \\ C_1 & \times & C_2 & \xrightarrow{f_C} & C_T \end{array}$$

(a) Overview of the modules and maps of Groth-Sahai proofs

$$\begin{array}{ccccc} \mathbb{G} & \times & \mathbb{G} & \xrightarrow{e} & \mathbb{G}_T \\ \mathtt{id} \downarrow & & \mathtt{id} \downarrow & & \mathtt{id} \downarrow \\ \mathbb{G} & \times & \mathbb{G} & \xrightarrow{e} & \mathbb{G}_T \\ \lambda \downarrow & & \lambda \downarrow & & \lambda \downarrow \\ \mathbb{G}_p & \times & \mathbb{G}_p & \xrightarrow{e} & \mathbb{G}_{T,p} \end{array}$$

(b) Instantiation for pairing product equations with SD-assumption

Figure 3.8: Groth-Sahai setup

to values in the A-modules are elements of the B-modules. A real *crs* will have $p_1(u_i) = 0$ and $p_2(v_j) = 0$ for all i and j. A proof is created by taking the desired equation on the commitments instead of the witnesses, and moving all randomness from the commitments to one side of the equation. This yields soundness, since by taking the p-maps to the C-modules, all this randomness will disappear, and the equation therefore holds for the values inside the commitments.

A simulation *crs* will have $\iota_1(A_1) \subseteq \langle u_1, ..., u_I \rangle$ and $\iota_2(A_2) \subseteq \langle v_1, ..., v_J \rangle$. This *crs* must be indistinguishable from a real *crs*. Commitments are now perfectly hiding, but not binding, as they can be opened any way the prover likes. However, every opening to commitments for which a proof can be created will yield the same proof. This shows that we have witness indistinguishability. By having one valid witness (e.g. for pairing product equations, when $t_T = 1$ we have witness $x_i = y_i = 1$), we can simulate proofs. Proofs can be simulated for all commitments (by opening them the right way) and the resulting proof will be equal to all other possible openings for those commitments, so we have zero knowledge.

[GS08] also describes how to proof multi-scalar multiplications and quadratic equations, and every type of equation can be proven using the SD assumption, the SXDH assumption or the DLIN assumption.

Chapter 4

Cryptographic Overview of Idemix

Idemix [CL01, CL03, CvH02, IBM13] (short for identity mixer) is an attribute-based credential system designed and implemented at IBM Zürich. In this section a cryptographic overview of Idemix is provided. A list of symbols and their meaning in Idemix is given in Table 4.1.

4.1 Users

Every Idemix user has a master secret m_1. Organizations know users by pseudonyms, which are commitments to m_1. This secret will also be the first attribute value of every credential of this user, which prevents users from sharing credentials without sharing the master secret. This also allows a user to prove that two different credentials both belong to him (and e.g. show that he is a Dutch student by showing that he has the Dutch nationality using one credential, and prove he is a student using a second credential), by proving that both credentials contain the same m_1.

Symbol	meaning
Γ	modulus of pseudonym group
ρ	order of pseudonym group
m_i	attribute values
l_m	bit length of attribute values
l_ϕ	constant that governs statistical zero-knowledge
l_e	bit length of e in CL-signatures
l'_e	size of interval e in CL-signatures is taken from
l_H	bit length of hash range used for Fiat-Shamir heuristic
A_h	indices of attributes hidden to the issuer during issuance
A_r	incides of attributes revealed while showing a credential
$A_{\bar{r}}$	indices of attributes hidden while showing a credential

Table 4.1: Idemix symbols and their meaning

4.1.1 Pseudonyms

Organizations know users by pseudonyms. The user chooses whether he will reuse a pseudonym and build up reputation with the organization, or use a new pseudonym every time and remain fully anonymous. Pseudonyms are computed in group $\langle g \rangle = \langle h \rangle$, a subgroup of $\mathbb{Z}_\Gamma^*$ of prime order ρ. Pseudonyms are Pedersen commitments to m_1: $\mathtt{Nym} = g^{m_1} h^u \pmod{\Gamma}$.

For some applications the organization must be sure that every user can only create one pseudonym with that organization, a domain pseudonym. For example, in an electronic voting scenario, eligible voters could have a credential that allow them to vote, but they should only be allowed to vote once. Since credentials can be shown unlinkably, a relying party cannot determine whether the voter has voted before. They can require the voters to show a domain pseudonym, which allows them to block a voter that tries to vote twice. Domain pseudonyms are computed by $\mathtt{DNym} = {g_{\mathtt{dom}}}^{m_1}$, with $g_{\mathtt{dom}} = \mathcal{H}(\mathtt{dom})^{(\Gamma-1)/\rho} \pmod{\Gamma}$, and $\mathcal{H}$ a hash function with range $\mathbb{Z}_\Gamma$, and $\mathtt{dom}$ an identifier of the domain.

4.2 Credentials

Idemix credentials are a CL-signature on a list of L attributes, with the user's master secret m_1 as first attribute of every credential. Credential issuers generate a key using the key generation process of the CL-signature. Messages have bit length l_m.

4.2.1 Issuance

To get a credential, the user must first convince the issuer that he is eligible for this credential. This can be done in many ways, face-to-face, or by showing other credentials the user already owns. After this, suppose a user known by pseudonym $\mathtt{Nym} = g^{m_1} h^u \pmod{\Gamma}$ wants a credential containing $m_1, ..., m_L$. He cannot simply send these values to the issuer, because m_1 is his personal secret, and perhaps the other attribute values contain private information as well. During the signing process, let A_h denote the indices of attributes that are not disclosed to the issuer. We never reveal m_1, so we always have $1 \in A_h$.

The user creates $U = S^{v'} \cdot \prod_{j \in A_h} R_j^{m_j} \pmod{n}$. The issuer cannot extract the message values from this commitment. The prover creates a proof that U is well formed, and that m_1 is used both in the credential and the secret in this pseudonym $\mathtt{Nym}$. Freshness of the proof is guaranteed by using a nonce generated by the issuer. The issuer checks the proof, and adds the other attribute values. Finally he creates the CL-signature, and proves knowledge of the secret exponent $e^{-1} \mod p'q'$. This proof's freshness is guaranteed by nonce n_2, chosen by the user. The user now receives the credential, which he can verify. The protocol is shown in Figure 4.1.

4.2.2 Showing Credentials

To achieve multi-show unlinkability, every time a prover shows his credential it must appear different. A prover can randomize signature (A, e, v) to (A', e, v') by taking random r_A, and computing $A' = A \cdot S^{r_A}$, and $v' = v - e r_A$. A prover cannot randomize e, as he needs the factorization of n to calculate $e^{-1} \pmod{\phi(n)}$. To prevent a verifier from linking multiple shows of a credential by recognizing the same e,

User $\{m_i\}_{i\in A_h}$	Public $\{m_i\}_{i\notin A_h}$	**Issuer** $n = pq = (2p'+1)(2q'+1)$
		$n_1 \in_R \{0,1\}^{l_\phi}$
	$\xleftarrow{n_1}$	
$v' \in_R \{0,1\}^{l_n+l_\phi}$		
$U = S^{v'} \cdot \prod_{j\in A_h} R_j^{m_j} \pmod n$		
$\sigma_1 = SPK\{(\{m_i\}_{i\in A_h}, v', r) :$		
$\quad U = \pm S^{v'} \prod_{j\in A_h} R_j^{m_j} \pmod n$		
$\quad \wedge nym = g^{m_1} h^r \pmod \Gamma$		
$\quad \wedge \bigwedge_{i\in A_h} m_i \in \{0,1\}^{l_m}\}(n_1)$		
$n_2 \in_R \{0,1\}^{l_\phi}$		
	$\xrightarrow{U,\sigma_1,n_2}$	
		verify σ_1
		prime $e \in_R [2^{l_e-1}, 2^{l_e-1} + 2^{l'_e-1}]$
		$v'' \in_R \{0,1\}^{l_v-1}$
		$Q = \frac{Z}{US^{v''} \prod_{i\notin A_h} R_i^{m_i}} \pmod n$
		$A = Q^{e^{-1} \pmod{p'q'}} \pmod n$
		$\sigma_2 = SPK\{(e^{-1}) :$
		$\quad A = \pm Q^{e^{-1}} \pmod n)\}(n_2)$
	$\xleftarrow{A,e,v'',\sigma_2}$	
$v = v' + v''$		
verify CL-signature (A, e, v)		
verify σ_2		

Figure 4.1: Credential Issuance Protocol

the prover cannot simply show e. By proving knowledge of e instead of revealing it, a verifier cannot recognize the same e being used multiple times. Finally, we need selective disclosure: the user must be able to choose which attributes will be shown to the verifier, and which remain private. Let A_r be the indices of attributes disclosed, and $A_{\bar{r}}$ the attributes that remain secret. The user proves knowledge of $\{m_i\}_{i\in A_{\bar{r}}}$, such that they are not shown to the verifier. He must also proof that the attribute values have the correct length, and that e is taken from the right interval. Formally he proves:

$$
\begin{aligned}
SPK\{&(e, \{m_i : i \in A_{\bar{r}}\}, v) : \\
&\frac{Z}{\prod_{i\in A_r} R_i^{m_i}} = A^e S^v \prod_{i\in A_{\bar{r}}} R_i^{m_i} \\
&\wedge \bigwedge_{i\in A_{\bar{r}}} m_i \in \{0,1\}^{l_m} \\
&\wedge e - 2^{l_e-1} \in \{0,1\}^{l'_e} \\
\}&
\end{aligned}
$$

The full proof is shown in Figure 4.2.

Prover $\frac{Z}{\prod_{i\in A_r} {R_i}^{m_i}} = A^e S^v \prod_{i\in A_{\bar{r}}} {R_i}^{m_i}$	Public n, R_i, S, Z	**Verifier** $\{m_i\}_{i\in A_r}$
$r_A \in_R \{0,1\}^{l_n+l_\phi}$ $A' = A \cdot S^{r_A}$ $v' = v - er_A$ $e' = e - 2^{l_e-1}$ $\widetilde{e} \in_R \pm\{0,1\}^{l'_e+l_\phi+l_H}$ $\widetilde{v'} \in_R \pm\{0,1\}^{l_v+l_\phi+l_H}$ $\widetilde{m_i} \in_R \pm\{0,1\}^{l_m+l_\phi+l_H}\ (i \in A_{\bar{r}})$ $\widetilde{Z} = {A'}^{\widetilde{e}} \left(\prod_{i\in A_{\bar{r}}} {R_i}^{\widetilde{m_i}}\right) S^{\widetilde{v'}}$		
	$\xrightarrow{A', \widetilde{Z}}$	
		$c \in_R \{0,1\}^{l_H}$
	$\xleftarrow{c}$	
$\widehat{e} = \widetilde{e'} + ce$ $\widehat{v'} = \widetilde{v'} + cv'$ $\widehat{m_i} = \widetilde{m_i} + cm_i\ (i \in A_{\bar{r}})$		
	$\xrightarrow{\widehat{e}, \widehat{v'}, \{\widehat{m_i}\}_{i\in A_{\bar{r}}}}$	
		${A'}^{\widehat{e}} S^{\widehat{v'}} \prod_{i\in A_{\bar{r}}} {R_i}^{\widehat{m_i}} \stackrel{?}{=}$ $\widetilde{Z} \left(\frac{Z}{{A'}^{2^{l_e-1}} \prod_{i\in A_r} {R_i}^{m_i}}\right)^c$ $\widehat{m_i} \in \pm\{0,1\}^{l_m+l_\phi+l_H+1}\ (i \in A_{\bar{r}})$ $\widehat{e} \in \pm\{0,1\}^{l'_e+l_\phi+l_H+1}$

Figure 4.2: The interactive ProveCL protocol

Chapter 5

Delegation of Idemix Credentials

In the previous chapters we've introduced many cryptographic primitives, proof techniques and described how Idemix works. In this chapter we use all these building blocks to present our main result: we propose an extension to Idemix which allows a single step of delegation. We start by defining the security requirements, after which we give a high level overview. Finally we present the full cryptographic construction, and analyze its effectiveness.

5.1 Security Requirements

Based on the delegation examples provided in the introduction (see Section 1.4), we have the following requirements on a delegation construction. Note that a formal security proof will not be provided, so we present the requirements in an informal way.

Correctness A delegated credential will always be accepted by an honest verifier.

Delegation evident A verifier can distinguish delegated credentials from non-delegated credentials.

Delegated selective disclosure A prover can selectively disclose attributes from a delegated credential.

Delegated multi-show unlinkability Delegated credentials are multi-show unlinkable.

Delegator unlinkability A delegator cannot link showings of a credential he delegated.

Delegator anonymity A verifier cannot link proofs of delegated credentials delegated by the same delegator.

Unforgeability An adversary has negligible probability of successfully proving he has a delegated credential which has not been delegated to him.

The need for these requirements should be trivial, except perhaps for delegation evidence. This is required for delegation to be usable in a practical way. When a verifier is unable to distinguish delegated from non-delegated credentials, a user could e.g. delegate his student credential, such his non-student friends also receive a student discount. However, a delegated student credential might be acceptable as a 'visitor pass' to access the university.

The issuer can prevent delegation of his credentials by refusing to hand out delegation credentials. In case of the student credential, we have solved the student discount problem, but also lost the functionality of university visitor passes. When a verifier can distinguish delegated from non-delegated credentials, the relying party can choose whether it makes sense to accept delegated credentials. The relying party that offers a student discount could reject delegated student credentials, while the university would still accept the visitor passes.

Another reason for delegation evidence is the driver's licence example covered in the introduction (Section 1.4). Driver's licences are delegated credentials of some sort, whereas the non-delegated version means you have the right to hand out driver's licences. In such a scenario, the non-delegated version has a different meaning that the delegated version, so a verifier must be able to distinguish the two.

5.2 High level overview

To allow delegation of credentials, we introduce a new type of credential, the *delegation credential*. For a credential the user owns, he can request a delegation credential. Such a credential contains the same attribute values $\{m_i\}_{1<i\leq L}$ as the normal credential, but it also contains the public key pk of a key pair chosen by the user. The intuitive meaning of this credential is 'whoever knows the secret key corresponding to public key pk has the right to allow someone to use the attribute values $\{m_i\}_{1<i\leq L}$'.

To delegate a credential, the delegator will create an autenticator. This authenticator is a signature on the delegatee's identity, using the secret key corresponding to the public key in the delegation credential. The delegatee receives the delegation credential including the attribute values and the public key, and the authenticator. Together this forms the *delegated* credential. Using the delegated credential, the delegatee can use attributes that the delegator has by showing the delegation credential ('the delegator has the right to delegate these attributes') and show the authenticator ('the delegator allowed me to use it').

The main challenge is that a delegated credential must still be multi-show unlinkble. Both the delegation credential and the authenticator must therefore shown unlinkably. A delegatee cannot simply show the public key by which the authenticator is verified, as showing this would make the delegatee (to some extend) linkable. We therefore need a signature scheme that allows us to unlinkably show the signature and public key, by e.g. randomization and proofs of knowledge, and prove that this is the public key in the delegation credential.

This construction differs from other delegation constructions (which we will cover in Chapter 6) in the sense that this construction involves delegation credentials. Why do we need this extra credential? Note that the user's secret m_1 is not part of the delegation credential. This is the main reason to introduce a new credential. A delegatee must prove that his delegator is authorized to delegate every time he

shows his delegated credential. He must do so unlinkably (as we require delegated multi-show unlinkability). Suppose we want to use the delegator's normal credential for this purpose. The delegator could give a non-interactive proof-of-knowledge of his this credential, but such a proof cannot be randomized by the delegatee, so he cannot show this unlinkably. In other constructions this is possible, as they use randomizable proofs. This is why they do not need delegation credentials. The delegatee cannot create proofs-of-knowledge of the delegator's normal credential either, because for this he needs the credential (A, e, v) and all the attribute values of the delegator, including his secret m_1, which the delegator cannot disclose. By introducing a delegation credential that does not contain secret values, the delegator can share all information on this credential with the delegatee, such that he can create unlinkable proofs-of-knowledge of it.

5.3 Construction

We now show how we instantiate the delegation credentials and authenticator.

5.3.1 Signature scheme for authenticator

We need a signature scheme that can be shown unlinkably, and that can be verified using a proof of knowledge of the public key. A plain RSA signature fulfills these requirements. The public key is (n, e), with n the product of primes p, q. A signature on m is σ such that $\sigma^e = m \pmod{n}$. Since (n, e) is the public key, we cannot show any of this. If e is the same for all public keys, then this is simply a system parameter and we only need to hide n in order to achieve unlinkability. We can do so by setting $e = 3$. The only requirement on e is that it is relatively prime with $(p-1)(q-1)$. A user can easily make sure this holds for $e = 3$, e.g. by taking two safe primes $p = 2p' + 1, q = 2q' + 1$. Now $(p-1)(q-1) = 4p'q'$, and when p', q' are primes greater than 3, $e = 3$ will be relatively prime to this.

We still need to keep the modulus secret, meaning we have to prove statements about modular arithmetic without revealing the modulus. Using the techniques described in Section 3.2.5, we can create such proofs. The number of modular exponentiations required for such proofs is linear in the bit length of the exponent, but by taking $e = 3$ this is still feasible, as shown in Section 3.2.5. Still, these proofs are computationally expensive. A signature scheme for which the modulus is not part of the public key (e.g. signature schemes in the discrete logarithm setting, where the group and modulus can be fixed for all key pairs) could possibly yield much more efficient constructions, but we know no such scheme that fulfills our unlinkability requirements. Suppose we use an ElGamal signature [ElG85]: (r, s) with $g^m = y^r r^s \pmod{p}$ where public key $y = g^x$. To have unlinkability, we would need to keep y secret as it is the public key, (r, s) because it is the signature and m, as this is some form of the user's identity. We know no efficient proof $PK\{(m, (r, s), y) : g^m = y^r r^s \pmod{p}\}$.

5.3.2 Hash function to mitigate existential forgery of authenticator

Many signature schemes are existentially forgeable, RSA is no exception. This forgery is problematic, as a user could forge an authenticator for some (non chosen)

master secret, and then start using this new master secret. As a result non-delegated users could use a delegated credential, if they manage to obtain the delegation credential.

Existential forgery can be mitigated by hashing the message before it is signed. However, hash functions destroy mathematical structure that we need in the proofs of knowledge. A solution is to assign one authority which issues 'hash credentials' containing a hash function of a domain pseudonym. Every user that wants to be able to receive delegated credentials must at some point go to this authority and receive a credential containing $h = \mathcal{H}(\texttt{DNym})$. The hash value h is bound to user secret m_1 as they are together in a credential. Now he can prove that he has a signature on $\mathcal{H}(\texttt{DNym})$ by showing that he has a signature on some h, and that this value is in the hash credential.

All hash credentials have the following form:

$$(A_{\mathcal{H}}, e_{\mathcal{H}}, v_{\mathcal{H}}) : {A_{\mathcal{H}}}^{e_{\mathcal{H}}} = \frac{Z_{\mathcal{H}}}{{S_{\mathcal{H}}}^{v_{\mathcal{H}}} {R_{\mathcal{H},1}}^{m_1} {R_{\mathcal{H},2}}^{\mathcal{H}(\texttt{DNym})}} \pmod{n_{\mathcal{H}}}$$

We would like to use a full domain hash or a probabilistic hash for this hash function, which results in provably secure RSA signatures (RSA-FDH and RSA-PSS). Unfortunately, this is not possible. We cannot use a full domain hash since such a hash function depends on the modulus of the public key the hash will be used with, and we want to use this hash value with multiple public keys (such that this value can be signed by multiple delegators). We cannot use a probabilistic hash to get RSA-PSS either, as this scheme defines that signing involves creating a new hash first. We cannot take new hash values because we want to store a single hash value in the hash credential.

We therefore introduce a new RSA signature that uses a half-domain hash (RSA-HDH). Let k be the bit lenght of the RSA modulus, we use a hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{k-1}$. The output of a half-domain hash can be signed using any RSA key with a k-bit modulus n, which implies $n > 2^{k-1}$. We can sign any value in $\mathbb{Z}_n$, and every element in $\{0,1\}^{k-1}$ is also element of $\mathbb{Z}_n$. We call this a half-domain hash, as it covers at least half of $\mathbb{Z}_n$: Since n is a k-bit RSA modulus, we know $n < 2^k$. $\mathcal{H}$ maps to 2^{k-1} distinct elements of $\mathbb{Z}_n$, and $2^{k-1} = \frac{1}{2}2^k$, so $\mathcal{H}$ covers more than half of $\mathbb{Z}_n$.

Theorem 1. *RSA-HDH is existentially unforgeable against an adaptive chosen message attack under the RSA assumption in the random oracle model.*

Proof. We prove this by reducing forgery of an RSA-HDH signature to breaking the RSA problem. This reduction is adapted from the security proof of RSA-FDH by Coron [Cor00], which is an improvement of the proof by Bellare and Rogaway [BR96].

We call RSA (t', ϵ')-broken if an attacker given some $y \in_R \mathbb{Z}_n^*$ and RSA public key (n, e) can with probability at least $\epsilon'(k)$ compute $y^{1/e} \pmod{n}$, in time $t'(k)$. Here k is the bit length of the modulus n. If no such attacker exists, we call it (t', ϵ')-secure. We call an RSA-HDH signature $(t, q_{\text{sig}}, q_{\text{hash}}, \epsilon)$-secure if no attacker that is allowed $q_{\text{hash}}(k)$ hash queries, $q_{\text{sig}}(k)$ signature queries can compute a forged signature on a new message m with probability at least $\epsilon(k)$ in running time $t(k)$, where k is the bit length of the modulus.

Suppose forger $\mathcal{F}$ $(t, q_{\text{sig}}, q_{\text{hash}}, \epsilon)$-forges RSA-HDH signatures. We create inverter $\mathcal{I}$ that executes $\mathcal{F}$ on his own (and answers all oracle queries $\mathcal{F}$ makes) and

Reduction	Security
RSA-FDH [BR96]	$\epsilon(k) = (q_{sig}(k) + q_{hash}(k) + 1) \cdot \epsilon'(k)$
	$t(k) = t'(k) - (q_{hash}(k) + q_{sig}(k) + 1) \cdot \mathcal{O}(k^3)$
RSA-FDH [Cor00]	$\epsilon(k) \cong q_{sig}(k) \cdot \epsilon'(k)$
	$t(k) = t'(k) - (q_{hash}(k) + q_{sig}(k) + 1) \cdot \mathcal{O}(k^3)$
RSA-HDH (this work)	$\epsilon(k) \cong q_{sig}(k) \cdot \epsilon'(k)$
	$t(k) = t'(k) - 2 \cdot (q_{hash}(k) + q_{sig}(k) + 1) \cdot \mathcal{O}(k^3)$

Table 5.1: Security of RSA signatures assuming RSA is (t', ϵ')-secure

breaks the RSA problem. $\mathcal{I}$ is given public key (n, e) and $y \in \mathbb{Z}_n^*$, and has to invert the RSA function on y and output $y^{1/e} \pmod n$.

$\mathcal{I}$ runs the forger and has to answer the hash queries. When forger makes a hash query on m_i, with probability $1 - \frac{1}{q_{sig}+1}$, $\mathcal{I}$ takes $r_i \in_R \mathbb{Z}_n$ and calculates $h_i = {r_i}^e \pmod n$. If this value is in $\{0,1\}^{k-1}$, $\mathcal{I}$ has created a valid oracle answer and returns h_i to $\mathcal{F}$. If not, $\mathcal{I}$ takes a new r_i and tries again. Since the half domain hash covers at least half of $\mathbb{Z}_n$, on average this takes less than 2 attempts. With probability $\frac{1}{q_{sig}+1}$ $\mathcal{I}$ takes $r_i \in_R \mathbb{Z}_n$ and calculates $h_i = y \cdot {r_i}^e \pmod n$. Again, with probability at least $\frac{1}{2}$ this is in $\{0,1\}^{k-1}$ and is a valid oracle answer. Otherwise $\mathcal{I}$ takes a new r_i and tries again, and on average less than 2 attempts are required.

When $\mathcal{F}$ makes a signing query on m_i, we assume he made the hash query on m_i before (and if not, $\mathcal{I}$ will create the hash query itself). $\mathcal{I}$ therefore knows the hash h_i corresponding to m_i, and with probability $1 - \frac{1}{q_{sig}+1}$ he has $h_i = {r_i}^e \pmod n$, so he can successfully answer the oracle query with r_i. With probability $\frac{1}{q_{sig}+1}$, $h_i = y \cdot {r_i}^e \pmod n$. In this case $\mathcal{I}$ cannot answer the signing query and fails.

Finally $\mathcal{F}$ outputs an attempted forgery σ on some message m_i. If he was successful, and corresponding hash $h_i = y \cdot {r_i}^e \pmod n$, $\mathcal{I}$ can compute $y^{1/e} \pmod n$ by taking $\frac{\sigma}{r_i} \pmod n$, and successfully inverted RSA. The probability that $\mathcal{F}$ succeeded is $\epsilon(k)$, the probability that $\mathcal{I}$ answers all signing queries correctly is $(1 - \frac{1}{q_{sig}+1})^{q_{sig}}$. The probability that $\mathcal{I}$ can extract $y^{1/e} \pmod n$ from forged signature σ is $\frac{1}{q_{sig}+1} = \frac{1}{q_{sig}} \cdot \frac{q_{sig}}{q_{sig}+1} = \frac{1}{q_{sig}} \cdot (1 - \frac{1}{q_{sig}+1})$. This means that $\mathcal{I}$ has probability $\epsilon'(k) = (1 - \frac{1}{q_{sig}+1})^{q_{sig}+1} \cdot \frac{1}{q_{sig}} \cdot \epsilon(k)$ of inverting RSA. For a large q_{sig}, we have $\epsilon'(k) \cong \frac{1}{q_{sig}} \cdot \epsilon(k)$. The time this takes is the running time of $\mathcal{F}$ and the time $\mathcal{I}$ needs to answer all queries. The time required to answer a query is mainly computing (on average) 2 modular exponentations, which takes $\mathcal{O}(k^3)$. This makes the running time of $\mathcal{I}$ equal to $t'(k) = t(k) + 2 \cdot (q_{sig} + q_{hash} + 1) \cdot \mathcal{O}(k^3)$. As shown in Table 5.1, this reduction is as tight as the original reduction for RSA-FDH by Coron [Cor00], and only differs in the time the attacker is given.

Existentially forging RSA-HDH signatures using an adaptive chosen message attack implies breaking the RSA problem, and we assume breaking the RSA problem to be infeasible, so the RSA-HDH signature is existentially unforgeable against an adaptive chosen message attack.

□

The output of this hash function is suitable for every RSA modulus of some fixed length. This allows one hash value to be signed by multiple delegators. Theorem 1 shows that the resulting signature scheme is provably secure.

5.3.3 Delegation credential

Delegation credentials will have the same form as other Idemix credentials. This means that an issuer now has to create two key pairs, one for normal credentials and one for delegation credentials. Let $n_1, S_1, Z_1, R_{1,1}, ..., R_{1,L}$ be the public key for normal credentials and $n_2, S_2, Z_2, R_{2,i}$ the public key for delegation credentials. The delegator has a credential with attribute values $\{m_i\}_{0<i\leq L}$: $(A_1, e_1, v_1) : {A_1}^{e_1} = \frac{Z_1}{{S_1}^{v_1} \prod_{i=1}^{L} {R_{1,i}}^{m_i}} \pmod{n_1}$. The delegation credential must contain the same attribute values. However, the delegatee must be able to show the delegation credential unlinkably. In order to prove knowledge of an Idemix credential, the prover must know A, e, v and all attributes $\{m_i\}_{0<i\leq L}$. Since m_1 is the delegator's master secret, revealing this to the delegatee is not possible. We therefore exclude m_1 from the delegation credential. We put a fresh public key n for the RSA signatures in its place. Note that the modulus must only be used in one delegation credential, such that authenticators are bound to the delegation credential. By reusing the modulus (and corresponding secret key), an authenticator might be used with a different delegation credential than intended. The resulting delegation credential will be $(A_2, e_2, v_2) : {A_2}^{e_2} = \frac{Z_2}{{S_2}^{v_2} {R_{2,1}}^{n} \prod_{i=2}^{L} {R_{2,i}}^{m_i}} \pmod{n_2}$.

5.3.4 Full construction

Suppose Alice wants to delegate a credential to Bob, and Bob has master secret m_1. Alice has delegation credential $(A_2, e_2, v_2) : {A_2}^{e_2} = \frac{Z_2}{{S_2}^{v_2} {R_{2,1}}^{n} \prod_{i=2}^{L} {R_{2,i}}^{m_i}} \pmod{n_2}$ with her public key n in it. Bob has hash credential $(A_{\mathcal{H}}, e_{\mathcal{H}}, v_{\mathcal{H}})$. Bob shows his hash credential and $h = \mathcal{H}(\texttt{DNym}_{Bob})$ to Alice, and Alice sends her public key n and places a RSA signature on h, giving Bob the authenticator σ. Bob also receives the delegation credential with attribute values.

To show subset A_r of the attributes from this delegated credential, Bob makes the following proofs.

$$
\begin{aligned}
PK\{(m_b, \{m_i\}_{i\in A_{\bar{r}}}, (A_2, e_2, v_2), (A_{\mathcal{H}}, e_{\mathcal{H}}, v_{\mathcal{H}}), h, n, \sigma)\} : \\
\frac{Z_2}{\prod_{i\in A_r} {R_{2,i}}^{m_i}} = {A_2}^{e_2} {S_2}^{v_2} {R_{2,1}}^{n} \prod_{i\in A_{\bar{r}}} {R_{2,i}}^{m_i} \quad \wedge \\
Z_{\mathcal{H}} = {A_{\mathcal{H}}}^{e_{\mathcal{H}}} {S_{\mathcal{H}}}^{v_{\mathcal{H}}} {R_{\mathcal{H},1}}^{m_1} {R_{\mathcal{H},2}}^{h} \quad \wedge \\
\sigma^3 = h \pmod{n}\}
\end{aligned}
$$

This proof is a conjunction of three parts. In the first part of the proof, the delegatee selectively discloses the delegation credential. Modulus n will never be disclosed. In the second part, he shows that h is his hash value by showing his hash credential. In the third part, he shows that he knows the authenticator, a signature on h signed by the owner of the delegation credential. The third part of the proof uses the secret modular arithmetic as described in Section 3.2.5. This technique internally proves statements about Pedersen commitments, so the prover discloses commitments $c_h = {g_1}^h {g_2}^{u_1}$ and $c_n = {g_1}^n {g_2}^{u_2}$. This allows the use of and-composition to enforce that the value signed is indeed h from the hash credential, and the modulus used is indeed the modulus from the delegation credential.

Task	Cost
Show delegation credential	$3 + L - \|A_r\|$
Show hash credential	5
Show authenticator	16
create commitments	4
prove commitments	8
multiplications	4
Total	$24 + L - \|A_r\|$

Table 5.2: The amount of modular exponentiations required to show a delegated credential

5.3.5 Efficiency Analysis

Let us analyze how efficient this construction is. The computational complexity is measured by counting the occurences of the most expensive operation, the modular exponentiation, that the prover must compute. We are interested in the complexity for the prover, because this involves secret keys and should be run on secure hardware such as a smart card, which often means limited computational power. The complexity is shown in Table 5.2. Showing an Idemix credential takes 2 modular exponentiations, plus one for every attribute that is not disclosed, because the prover must prove knowledge of it. The signature must also be randomized, which takes one exponentiation. This is equal to $3 + L - |A_r|$. This also holds for the hash credential, but we know that this credential has only 2 attribute values, both of which will remain hidden, so the cost is $3 + 2 = 5$. Showing the authenticator consists of creating the required commitments, and proving that the values in the commitments have the desired relation. We need to commit to four values: σ, h, n, λ. Creating such a commitment takes only one modular exponentiation, if we precompute the values of the g-generator and only add the randomness with the h-generator. For four commitments this will take 4 exponentiations. Proving the representation of the commitment takes 2 modular exponentiations, making a total of 8 exponentiations for the four commitments. The last step is to prove that $\sigma^2 = \lambda \pmod{n}$ and $\sigma \cdot \lambda = h \pmod{n}$. These secret multiplications cost 2 each, adding 4 modular exponentiations to the total. All steps combined cost $24 + L - |A_r|$ modular exponentiations.

Note that not all modular exponentiation are equally expensive, because the modulus size differs. The issuer must use different parameters for delegation credentials, as the message length l_m must be large enough to hold an RSA modulus. For security, l_e (the length of exponents e used in CL-signatures) must be larger than $l_\phi + l_H + l_m + 4$, which requires the RSA modulus of the issuer to be even larger. This makes the modular exponentiations of showing the delegation credential and hash credential more expensive than the others, and means that selective disclosure of delegated credentials will result in a bigger performance hit than selective disclosure of normal credentials. The secret modular arithmetic requires the group order to have a bit length of more than twice the maximum bit length of the numbers of which the prover proves knowledge. All these modular exponentiations are therefore more expensive than the modular exponentiations of showing a normal Idemix credential.

Let us consider a very rough estimate of the computation time this would take

on a smart card. Showing an Idemix credential that contains five attributes without disclosing any attribute can be done in 1.5 seconds on a smart card, and a single modular exponentiation takes 100ms, both using a 1024 bit modulus [VA13]. Showing a credential that contains only 2 attributes, without disclosing either, takes 1.1 second. Suppose we use a 2048 bit RSA modulus for delegation credentials and hash credentials (to allow a sufficiently large message space for the public keys and the hash values), and suppose doubling the modulus doubles the computation time. This would mean showing a delegation credential takes 2.2 seconds, showing a delegation credential with 5 attributes would take between 1.9 and 2.7 seconds (depending on the amount of attributes disclosed). If we also use a group with a modulus of 2048 bits to show the authenticator, and assume that modular exponentiations now take 200ms, the modular exponentiations alone take 3.2 seconds. Adding time for other computation and communication, showing the authenticator might take 5 seconds. This very rough estimate suggests that it takes about 10 seconds to show a delegated credential. For many applications (e.g. using a credential as a public transport ticket), 10 seconds is way too slow. However, it is close to being practical. With progress on smart card technology and an efficent implementation we can imagine this number decreasing to a couple of seconds, which would suffice for some applications.

5.4 Security Analysis

We now analyze the security properties the proposed construction satisfies. Here we assume that users do not share their user secrets m_1, because sharing m_1 allows sharing of delegated credentials (and non-delegated credentials too). We therefore consider knowing a user secret m_1 as being the owner of m_1. We also assume that delegators follow the protocol and use a new public key for each delegation credential. We assume that that the issuer only issues delegation credentials to users that are allowed to delegate, and the hash authority only issues credentials with attributes $m_1, \mathcal{H}(\texttt{DNym})$ with DNym a domain pseudonym of user with secret m_1. Finally we assume that the attribute values are not identifying and do not reveal the identity of the user or delegator.

Lemma 1. *A prover that has non-negligible probability of successfully proving knowledge of a delegated credential must know a delegated credential* $\{m_i\}_{0<i\leq L}, (A_2, e_2, v_2), (A_{\mathcal{H}}, e_{\mathcal{H}}, v_{\mathcal{H}}), h, n, \sigma$, *with* $Z_2 = {A_2}^{e_2} {S_2}^{v_2} {R_{2,1}}^{n} \prod_{1<i\leq L} {R_{2,i}}^{m_i}$, $Z_{\mathcal{H}} = {A_{\mathcal{H}}}^{e_{\mathcal{H}}} {S_{\mathcal{H}}}^{v_{\mathcal{H}}} {R_{\mathcal{H},1}}^{m_1} {R_{\mathcal{H},2}}^{h}$, *and* $\sigma^3 = h \pmod{n}$.

Proof. This follows directly from the knowledge soundness property of the proofs-of-knowledge. □

Lemma 2. *A user with secret m_1 can only get one h-value with valid hash credential $(A_{\mathcal{H}}, e_{\mathcal{H}}, v_{\mathcal{H}})$ such that $Z_{\mathcal{H}} = {A_{\mathcal{H}}}^{e_{\mathcal{H}}} {S_{\mathcal{H}}}^{v_{\mathcal{H}}} {R_{\mathcal{H},1}}^{m_1} {R_{\mathcal{H},2}}^{h}$.*

Proof. By unforgeability of the CL-signature, only the hash authority can create hash credentials. In order to receive a hash credential, the user must give his domain pseudonym, which is a function of his secret m_1. A user can only create a single DNym with m_1, and the hash authority will only issue $h = \mathcal{H}(\texttt{DNym})$ (by the assumption that the hash authority is honest). □

We now describe how this construction satisfies the requirements given in Section 5.1.

Correctness

Suppose a user has hash credential $(A_{\mathcal{H}}, e_{\mathcal{H}}, v_{\mathcal{H}})$ with ${A_{\mathcal{H}}}^{e_{\mathcal{H}}} = \frac{Z_{\mathcal{H}}}{{S_{\mathcal{H}}}^{v_{\mathcal{H}}} {R_{\mathcal{H},1}}^{m_1} {R_{\mathcal{H},2}}^{\mathcal{H}(\texttt{DNym})}} \pmod{n_{\mathcal{H}}}$. He knows his secret m_1 and h. When a delegator delegates to this user, he receives a delegation credential (A_2, e_2, v_2) with ${A_2}^{e_2} = \frac{Z_2}{{S_2}^{v_2} {R_{2,1}}^{n} \prod_{i=2}^{L} {R_{2,i}}^{m_i}} \pmod{n_2}$, attribute values $\{m_i\}$, n and authenticator σ with $\sigma^3 = h$. All these values together form a witness for the relation of the proof for showing a delegated credential. By completeness of the proofs, an honest verifier will accept this delegated credential.

Delegation evident

Delegated credentials clearly differ from non-delegated credentials and use a different proof protocol. A verifier can therefore easily distinguish the two.

Delegated selective disclosure

The attribute values come from the delegator credential, and this is a standard Idemix credential. Selective disclosure for delegated credentials therefore works just like it works for other Idemix credentials.

Delegated multi-show unlinkability

When showing a delegated credential, the user shows the delegator credential, his hash credential, and proves knowledge of the authenticator. The first two are Idemix credentials, and these are multi-show unlinkable. Since we create a zero-knowledge proof of knowledge of the authenticator (and only reveal freshly randomized commitments), the combination is also multi-show unlinkable.

Delegator unlinkability

The delegator knows the delegation credential and the authenticator. However, he is not able to recognize uses of this delegated credential, due to the witness indistinguishability of the proofs. He is unable to recognize the authenticator, because the prover creates a zero-knowledge proof of knowledge, and only freshly randomized commitments are revealed.

Delegator anonymity

The difference between two delegation credentials (other than the attribute values) is the public key n it contains, and consequently the public key under which the authenticator is valid. A verifier will not learn anything about n, since the prover only proves knowledge of it. It is not disclosed from the delegation credential, and only a fresh commitment to n is revealed while showing the authenticator.

Unforgeability

By Lemma 1, a prover with non-negligible probability of successfully proving knowledge of a delegated credential must know $\{m_i\}_{0<i\leq L}$, (A_2, e_2, v_2), $(A_{\mathcal{H}}, e_{\mathcal{H}}, v_{\mathcal{H}})$, h, n, σ, with $Z_2 = {A_2}^{e_2} {S_2}^{v_2} {R_{2,1}}^{n} \prod_{i=2}^{L} {R_{2,i}}^{m_i}$, $Z_{\mathcal{H}} = {A_{\mathcal{H}}}^{e_{\mathcal{H}}} {S_{\mathcal{H}}}^{v_{\mathcal{H}}} {R_{\mathcal{H},1}}^{m_b} {R_{\mathcal{H},2}}^{h}$, and $\sigma^3 = h \pmod{n}$.

By unforgeability of the CL-signature, the delegation credential has to be issued by the issuer and the owner of public key n is indeed authorized to delegate (since we trust the issuer). The adversary is therefore unable to forge

a delegation credential with his own public key. The adversary may however obtain valid delegation credentials with public keys $\{n_i\}$. By Lemma 2, a user with secret m_1 can only obtain a single hash credential, which contains some hash value h. An adversary can use many user secrets $\{m_{1,i}\}$, and for each get a hash credential with hash value $\{h_i\}$, and he needs a signature valid under some public key in $\{n_i\}$ on some h_i. By unforgeability of RSA-HDH (Theorem 1), signatures can only be created by the owner of the public key that knows the secret factorization. The owner of the delegation credential is the only one that knows the factorization of n, so only he can delegate using his delegation credential. Because the adversary does not know this for any n_i, he cannot create such a signature.

The adversary cannot use the authenticator of a different user, as the probability that this signature is on h' which is any of the h_i values is negligible. The adversary cannot use an authenticator of some other delegation credential, because the public key n' of this signature is not n (as we require delegators to use a new RSA modulus for every delegation credential).

Since the adversary cannot create an authenticator and cannot use one intended for someone else, he does not know a valid delegated credential and has negligible probability of convincing a verifier he does.

Chapter 6

Related Work

Chase and Lysyanskaya were the first to construct delegatable credentials [CL06]. In order to do so, they first define signatures of knowledge. They instantiate these using general NIZK proofs (authors suggest [Sah99] and [dSdCO+01]) and use NP reduction, which makes this construction quite inefficient. Moreover, credentials in this construction have exponential size in the delegation level. As the authors state, this construction is not efficient enough to be used in practice. Belinky et al. created delegatable credentials from randomizable proofs, in which credentials grew linearly in the delegation level [BCC+09]. Chase et al. created a similar construction, but based on malleable signatures instead of randomizable proofs [CKLM13]. Since the constructions by Belinky et al. and Chase et al. are more efficient, we discuss these in a bit more detail.

6.1 Delegatable Credentials from Randomizable Proofs

Belenkiy et al. start by showing that Groth-Sahai proofs [GS08] are randomizable: let π be a proof and $c_1, ..., c_n$ the commitments. Now anyone can randomize commitment $c_i = Commit(x_i, open_i)$ to $c_i' = Commit(x_i, open_i')$, and create π' such that this is a valid proof again. Note that this can be done without knowing the witness $\{x_i\}$ or the openings to the commitments. The delegatable credentials heavily rely on these randomizable proofs.

Every user U has a secret key sk_U. Pseudonyms, Groth-Sahai commitments to the user's secret key, are also used as public keys. There is no difference between users and authorities. To become an authority, a user simply publishes a public key.

Suppose an authority O issues a credential to user A, known to him by $\mathtt{Nym}_A$. This is called a level 1 credential, as it comes directly from an issuer. The issuer O creates a signature on sk_A using his secret key. Secure two party computation is used to create this signature, such that O will not learn sk_A (Fuchsbauer built upon this work which allows non-interactive issuing [Fuc10]). The signature scheme used is a combination of multiple weak Boneh-Boyen signatures. These signatures can be represented as a pairing product equation, which allows the use of Groth-Sahai proofs of knowledge of a signature. The authority O does not send the signature to A, but instead sends a NIZK proof of knowledge of this signature, called π_A. Now $\mathtt{Nym}_A$ and π_A form A's credential.

To use this credential, A cannot simply show $\mathtt{Nym}_A$ and π_A every time, as this

would be linkable. To prevent this, A first randomizes $\mathtt{Nym}_A$ and π_A using the randomizability of Groth-Sahai proofs, such that multi-show unlinkability is guaranteed.

To delegate her credential to B with pseudonym $\mathtt{Nym}_B$, A does the same thing the issuer did: she signs $\mathtt{Nym}_B$ using her secret key, and creates a NIZK proof of knowledge π_B of this signature. She sends π_B to B, together with her own (randomized) credential $(\mathtt{Nym}'_A, \pi'_A)$. Now B has a proof that $\mathtt{Nym}_A$ has a level 1 credential from issuer O, and a proof that she delegates this credential to him, giving him a level 2 credential. To show his credential, B must not only randomize $\mathtt{Nym}_B$ and π_B, but also randomize $(\mathtt{Nym}'_A, \pi'_A)$. He can do this, since no knowledge about the secret key or witness is needed to do so.

Generally, a user with a level L credential can issue a level $L+1$ credential by creating a signature on the receiver's pseudonym and sending a proof-of-knowledge of this signature, and by showing that he has a level L credential. To show this level L credential, the entire credential chain must be randomized every time. Since everything can be randomized, multi-show unlinkability and delegator unlinkability hold. The size of a credential grows linearly in the level of the credential, because every delegation adds one proof. This means that it is practical to have long delegation chains.

The signature scheme used is based on the weak Boneh-Boyen signature scheme [BB04]. This signature cannot be used as-is. A signature using signing key sk_A on someone else's key sk_B is equal to a signature using key sk_B on sk_A. Also, we want to sign a list of messages, and the standard signature would require a larger group if more messages must be signed. To solve these problems, the signer first signs a temporal key K^*. Using this key, the signer can sign multiple other temporal keys K_i. Now using keys K_i a list of message m_i can be signed. The F-unforgeable way (as described in Section 2.7.3) is used, with $F(m) = (h^m, u^m)$. This gives the following way to sign a list of messages: $Auth(params, sk, m_1, ..., m_L) = (A^* = g^{\frac{1}{sk+K^*}}, B^* = h^{K^*}, C^* = u^{*K*}, \{A_i = g^{\frac{1}{K^*+K_i}}, B_i = h^{K_i}, C_i = u_i^{K_i}, D_i = g^{\frac{1}{K_i+m_i}}\}_{1\leq i\leq L})$. Note that different u-values in the F function are being used. This prevents an attacker from changing the order of the messages.

Let $(A^*, B^*, C^*, \{A_i, B_i, C_i, D_i\})$ be the signature. Verification is done by checking

- $e(A^*, h^{sk}B^*) \stackrel{?}{=} e(g,h)$
- $e(B^*, u^*) \stackrel{?}{=} e(h, C^*)$
- $e(A_i, B^*B_i) \stackrel{?}{=} e(g,h)$
- $e(B_i, u_i) \stackrel{?}{=} e(h, C_i)$
- $e(D_i, B_i h^{m_i}) \stackrel{?}{=} e(g,h)$

All these statements be proved using Groth-Sahai proofs. This construction of this signature also allows selective disclosure.

6.2 Delegatable Credentials from Malleable Signatures

A malleable signature scheme allows one given message m and signature σ to create a signature σ' on message $T(m)$, where T is some allowable transformation. The

transformation is context hiding, which means that σ' does not reveal anything about the original message m.

Again every user has a secret key, and pseudonyms are commitments to this secret. An issuer is simply a user that published his public key. Credentials are signatures under the public key of an issuer, on messages of the following form: $m = (nym, l, flag)$. The owner's pseudonym is nym, the level is l. The flag is either `credential` or `proof`.

Suppose an issuer I issues a credential to A with pseudonym nym_A. I will create a signature on $(nym_A, 1, \texttt{credential})$. To show a credential, a user is required to show a signature on $(m, l, \texttt{proof})$. Note that we now need the flag `proof`, whereas the user has signature on flag `credential`. Luckily, the signature the issuer places is malleable. We have an allowed transformation $T(nym, l, \texttt{credential}) = (nym', l, \texttt{proof})$ where nym and nym' are pseudonyms of the same user. To use this transformation, the user must know $sk, open, open'$ with $nym = Commit(sk, open)$ and $nym' = Commit(sk, open')$. This allows the owner of the pseudonym (i.e. the person that knows the opening) to prove he owns the credential. Note that he can change the pseudonym, so he can also proof possession of the credential to a relying party that knows him nym'', by transforming the signature to a statement on nym''. By the context hiding property, this is provides unlinkability.

Delegation involves another transformation. Let user A with nym $nym_A = Commit(sk_A, open_A)$ have a level l credential. Using sk_A and $open_A$, he can use transformation $T(nym_A, l, \texttt{credential}) = (nym_B, l+1, \texttt{credential})$. When using this credential, one cannot learn anything about the delegator, again by the context hiding property.

Chase et al. continue give an instantiation of malleable proofs from NIZK proofs. Groth-Sahai proofs can be used for this instantiation.

Chapter 7

Conclusion

Attribute-based credentials can greatly protect users' privacy, by limiting the information disclosed and doing so unlinkably. The ability to delegate such credentials would allow even more use cases to be executed in a privacy-friendly way. We have constructed a way that allows one step delegation of Idemix credentials. This enables many new use cases for attribute-based credentials. Our construction is efficient, as it does not require pairings, and only uses a limited number modular exponentiations. We estimate that showing a delegated credential on current smart cards takes 10 seconds. It is an extension, as it can be added to Idemix to offer extra functionality, but nothing will change for users that do not want to use delegation. This would also simplify the introduction of delegation to existing Idemix systems. In addition, we have shown how the RSA-FDH construction can be used with multiple moduli by introducing the half-domain hash, and proved RSA-HDH to be secure in the random oracle model.

For future work, we obviously need an implementation of this construction. This would also allow to measure the efficiency and verify our estimated timings. The Idemix proofspec needs to be extended such that it allows proofs of delegated credentials. Our delegation construction leaves room for improvement. It could be improved by preventing users from sharing delegation keys, e.g. by using some form of revocable privacy, such that publishing your public key means sharing your secret key. The addition of selective delegation (delegate only certain attributes of a credential) and multiple levels of delegation (delegation of delegated credentials) to this construction are other improvements that would make this more useful. Finally, this construction could be made more efficient if one can find a signature scheme for the authenticator that fulfills the unlinkability requirements, without relying on the inefficient secret modular arithmetic. A signature scheme that is secure without a hash function would remove the need for a hashing authority.

Bibliography

[ACHdM05] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, 2005.

[ADR02] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In LarsR. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 83–107. Springer Berlin Heidelberg, 2002.

[AHS13] Gergely Alpár, Jaap-Henk Hoepman, and Johanneke Siljee. The identity crisis - security, privacy and usability issues in identity management. *Journal of Information System Security*, 9(1):23–53, 2013.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer Berlin Heidelberg, 2004.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, 2008.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matt Franklin, editor, *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer Berlin Heidelberg, 2004.

[BCC+09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 108–125. Springer Berlin Heidelberg, 2009.

[BCKL08] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *Theory of Cryptography*, volume 4948 of *Lecture Notes in Computer Science*, pages 356–374. Springer Berlin Heidelberg, 2008.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual*

ACM Symposium on Theory of Computing, STOC '88, pages 103–112, New York, NY, USA, 1988. ACM.

[BG93] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In ErnestF. Brickell, editor, *Advances in Cryptology - CRYPTO 92*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer Berlin Heidelberg, 1993.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In Joe Kilian, editor, *Theory of Cryptography*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer Berlin Heidelberg, 2005.

[Blu83] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, January 1983.

[Bon98] Dan Boneh. The decision diffie-hellman problem. In JoeP. Buhler, editor, *Algorithmic Number Theory*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer Berlin Heidelberg, 1998.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In Ueli Maurer, editor, *Advances in Cryptology - EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer Berlin Heidelberg, 1996.

[Bra99] Stefan Brands. *Rethinking public key infrastructures and digital certificates: building in privacy*. PhD thesis, Eindhoven University of Technology, 1999.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo G. Desmedt, editor, *Advances in Cryptology - CRYPTO 94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer Berlin Heidelberg, 1994.

[CE87] David Chaum and Jan-Hendrik Evertse. A secure and privacy-protecting protocol for transmitting personal information between organizations. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO 86*, volume 263 of *Lecture Notes in Computer Science*, pages 118–167. Springer Berlin Heidelberg, 1987.

[CFT98] Agnes Chan, Yair Frankel, and Yiannis Tsiounis. Easy come - easy go divisible cash. In Kaisa Nyberg, editor, *Advances in Cryptology - EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 561–575. Springer Berlin Heidelberg, 1998.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, October 1985.

[CKLM13] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: Complex unary transformations and delegatable anonymous credentials. Cryptology ePrint Archive, Report 2013/179, 2013.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology Ů EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer Berlin Heidelberg, 2001.

[CL03] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi, editors, *Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer Berlin Heidelberg, 2003.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matt Franklin, editor, *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer Berlin Heidelberg, 2004.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 78–96. Springer Berlin Heidelberg, 2006.

[CL11] Melissa Chase and Kristin Lauter. An anonymous health care system. Cryptology ePrint Archive, Report 2011/016, 2011.

[CM99] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, pages 107–122. Springer Berlin Heidelberg, 1999.

[Cor00] Jean-Sebastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer Berlin Heidelberg, 2000.

[CP93] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO 92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer Berlin Heidelberg, 1993.

[Cra97] Ronald Cramer. *Modular design of secure yet practical cryptographic protocols*. PhD thesis, CWI & University of Amsterdam, 1997.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In Burton S. Kaliski Jr, editor, *Advances in Cryptology - CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer Berlin Heidelberg, 1997.

[CS02] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. Cryptology ePrint Archive, Report 2002/161, 2002.

[CvH02] Jan Camenisch and Els van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 21–30, New York, NY, USA, 2002. ACM.

[Dam90] Ivan Damgård. Payment systems and credential mechanisms with provable security against abuse by individuals. In Shafi Goldwasser, editor, *Advances in Cryptology - CRYPTO 88*, volume 403 of *Lecture Notes in Computer Science*, pages 328–335. Springer New York, 1990.

[Dam02] Ivan Damgård. On σ-protocols. 2002. `http://www.cs.au.dk/~ivan/Sigma.pdf`.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, Nov 1976.

[dSdCO+01] Alfredo de Santis, Giovanni di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 566–598. Springer Berlin Heidelberg, 2001.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In GeorgeRobert Blakley and David Chaum, editors, *Advances in Cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer Berlin Heidelberg, 1985.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO 86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer Berlin Heidelberg, 1987.

[Fuc10] Georg Fuchsbauer. Commuting signatures and verifiable encryption and an application to non-interactively delegatable credentials. Cryptology ePrint Archive, Report 2010/233, 2010.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, April 1988.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 1994.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 339–358. Springer Berlin Heidelberg, 2006.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11:1–11:35, June 2012.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel Smart, editor, *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer Berlin Heidelberg, 2008.

[HGS99] Nicholas Howgrave-Graham and Jean-Pierre Seifert. Extending wiener's attack in the presence of many decrypting exponents. In *Secure Networking - CQRE [Secure] '99*, volume 1740 of *Lecture Notes in Computer Science*, pages 153–166. Springer Berlin Heidelberg, 1999.

[IBM13] IBM. Specification of the identity mixer cryptographic library v2.3.40. Technical report, IBM Research Zurich, 2013.

[JK03] Jakob Jonsson and Burt Kaliski. Public-key cryptography standards (pkcs)# 1: Rsa cryptography specifications version 2.1. 2003.

[LRSW00] Anna Lysyanskaya, RonaldL. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard Heys and Carlisle Adams, editors, *Selected Areas in Cryptography*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer Berlin Heidelberg, 2000.

[Oka93] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In ErnestF. Brickell, editor, *Advances in Cryptology - CRYPTO 92*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer Berlin Heidelberg, 1993.

[Paq13] Christian Paquin. U-prove technology overview v1.1 revision 2. Technical report, Microsoft Technical Report, 2013.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO 91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Berlin Heidelberg, 1992.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli Maurer, editor, *Advances in Cryptology - EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer Berlin Heidelberg, 1996.

[QQQ+90] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, MichaÃńl Quisquater, Louis Guillou, MarieAnnick Guillou, GaÃŕd Guillou, Anna Guillou, GwenolÃľ Guillou, and Soazig Guillou. How

to explain zero-knowledge protocols to your children. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 628–631. Springer New York, 1990.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 543–553, 1999.

[Sch91] C.P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[SM10] Santanu Sarkar and Subhamoy Maitra. Cryptanalysis of RSA with more than one decryption exponent. *Information Processing Letters*, 110(8-9):336 – 340, 2010.

[VA13] Pim Vullers and Gergely Alpár. Efficient selective disclosure on smart cards using idemix. In Simone Fischer-Hübner, Elisabeth Leeuw, and Chris Mitchell, editors, *Policies and Research in Identity Management*, volume 396 of *IFIP Advances in Information and Communication Technology*, pages 53–67. Springer Berlin Heidelberg, 2013.