university of groningen / faculty of mathematics and natural sciences / mathematics

Smart card integration in the pseudonym system *idemix*

Luuk Danes

Master's Thesis
18 December 2007

TNO Information and Communication Technology

Supervisor at the University of Groningen
Prof. dr. Jaap Top

Supervisor at TNO Information and Communication Technology
Dr. Jaap-Henk Hoepman

Co-supervisor at the University of Groningen
Prof. dr. Ruth F. Curtain

Corrected version — 10 January 2008

"Du kannst jemanden, der die Augen verbunden hat, noch so sehr aufmuntern, durch das Tuch zu starren, er wird doch niemals etwas sehen; erst wenn man ihm das Tuch abnimmt, kann er sehen."

Franz Kafka, Das Schloß
published posthumously in 1926

Preface

This thesis is the last project of my study of Mathematics at the University of Groningen. It is also the final result of an internship I performed with the Security Group of TNO ICT.

Hereby I would like to thank my supervisor at TNO, Jaap-Henk Hoepman, for his help, relevant comments and patience.

I would also like to thank my supervisor, study advisor and teacher Jaap Top. He has not only helped and supported me writing this thesis, but has been a great study advisor and teacher over the past few years. Especially the last year, which was tough for me, I have received a lot of support and encouragement from him.

I have to thank my friends for their support and their interest in me as a person and in my work. Just drinking coffee with them already makes me feel happy!

And of course, I have to thank my parents and my family for their continuous support and encouragement throughout the years.

Contents

Chapter 1

Introduction

Communication and commerce are shifting from the physical world into the digital world. Because a lot of digital service require personal information, our identity information is scattered throughout many computer systems.

Companies create databases with customer information to organise and optimise their transactions. Banks give their customers access to electronic banking web sites. Governments also provide online services to companies and citizens.

Such web based services need to know who is accessing them in order to personalise the experience or to present the correct personal information. Moreover, privacy protection is a real concern.

Consequently, citizens, governments and companies have a strong need for identity management — management of (online) identity information.

The first sections of this chapter are an introduction to (federated) identity management, and show a privacy friendly view on it by means of using smart cards in *idemix*. Sections 1.7 and 1.8 contain the main research question and the outline of this thesis.

1.1 Identity Management

Management of (online) identity information, or in short, identity management, is becoming increasingly important in our society.

Unfortunately, there is no unique definition of identity management, but a definition that works for the context of this thesis is as follows:

Definition 1.1.1 (Identity Management (IM))**.**
Identity management *is the set of processes and techniques to handle identity information for authentication and authorisation.*

In this definition, *authentication* is the verification of a (digital) identity and *authorisation* is the verification whether someone is allowed to undertake a specific action. What people expect identity management (IM) to do depends on who it uses. We will distinguish users, companies and governmental organisations.

For **users**, identity management will mostly be about convenience and privacy. Nowadays users have a lot of username-password pairs on the internet, for convenience it would be better if this amount could be reduced. To handle so many usernames and passwords, a user will likely choose the same username and password on different sites and will also choose easy to remember passwords.

For users there are also privacy concerns. On websites and in other ways companies demand a lot of identity information from the users. This certainly has benefits for the user, as profiling and thus personalisation of services is possible. On the other hand, the user has no control over what is done with that information. In the Netherlands data collection is restricted by law, and although legal actions are possible if a company or the government has abused your identity information, there is no active inspection on the companies. In other countries the use of identity information is often less controlled.

For **companies**, identity management is interesting from the point of view of information security and efficiency. Identity management has multiple purposes for companies. Companies provide products or services, and often have a relationship with their customers (which can be persons or companies) and there is need to identify and keep record of these customers.

Firstly, it is necessary to identify the customer when the company contacts the customer or when the customer contacts the company. This contact can be in a digital way, by entering a username and password on the company's website or a digital ordering system, but also more physical, such as by showing an identity card to enter a restricted area or by signing a receipt for packet delivery.

Secondly, managing the identity information of customers in a proper way increases efficiency for a company, as the company can handle orders more quickly and accurately. A company that can combine different sources of information about customers can profile them and give better service to them. As mentioned earlier, this at the same time raises privacy issues for the customer.

In addition, identity management is used internally as a tool to manage authorisations for employees. This avoids the necessity of having several usernames and passwords to access buildings and (online) applications.

Governments are interested in identity management in different ways. The government is a provider of identity documents such as passports, identity cards and driving licenses, but the government also issues identifiers as social security numbers (or comparable in the Netherlands: sofi-nummers).

Simultaneously, the government provides services to their citizens, to companies and to other organisations for which this identifying information is needed. To citizens this can be for example the granting of official documents such as passports, parking licenses or tax return forms. Companies have to be registered for tax collection or to grant permissions. Moreover, the government has to make laws about the collection of identity information and has to regulate it.

To describe identity management we define three different types of participants: *users*, *identity providers* and *service providers*. A user is 'someone' (which can be a human or a company) who seeks/requires a service. First, authentication and authorisation is done by an identity provider. And when the user is authorised, the service is given by the service provider.

In simple (non-federated) identity management, identities are always managed within the same domain as the service is given.

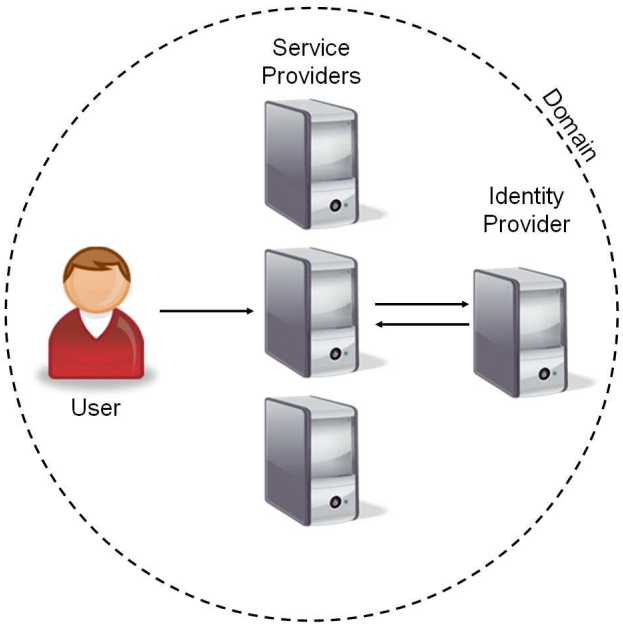

Figure 1.1: *Simple Identity Management.* A user wants a service from the service provider. The identity provider in the same domain is involved to authenticate the identity of the user.

1.2 Federation

At the moment, much attention is paid to Federated Identity Management, i.e., identity management spread over multiple domains. These domains can be either inside or outside the same organisation.

Within an organisation, federation is used to link different systems that are needed in processing or for single sign-on applications. Identity management that crosses the borders of organisations is used to optimise processes in which multiple organisations are involved.

Until now there has not been much attention for federation, and the existing solutions for federated identity management are mostly technical. Within organisations, federated identity management is already used, but there are almost no solutions for federation between different organisations.

For federation over different organisations trust and a set of communication rules are necessary. To accomplish this, a Circle of Trust has to be set up. Such a circle consists of service providers, identity providers and users. These providers and users need to trust each other. For users this means that their privacy is guaranteed and that their information will not be misused. Service providers want to be sure that the identity providers are reliable and that the identities they provide are correct and that the users are correctly authenticated.

A good example of federated identity management is Microsoft Passport. With Microsoft Passport single sign-on on the chat service MSN messenger and Microsoft web applications such as the email-service Hotmail and weblog/photo album-service Live Spaces is made possible. The use of this (federated) identity management system gives convenience to the users, as they need to log in only once. It was also the aim of Microsoft that other service-providers on the internet would use this system to identify their users, but a lack of trust of the providers with respect to Microsoft worked against this. The Circle of Trust could not be established by lack of trust. In retrospect we can conclude that Microsoft Passport has been too much about technology, and too little about trust.

Presently, a lot of parties are interested in becoming an identity provider. It is possible that governments will be identity providers, as they have always been and because they issue identification documents as driving licenses and passports. But also banks and even telecom providers are in the race. Banks have a relationship of trust with their customers for years and are seen as reliable. Telecom providers have less trust in their relationship with their customers, but because they are very much involved in communica-

tion between users and service providers, they nevertheless are interesting as identity provider. Moreover, they have the technology (mobile phone & SIM card) to authenticate their users reliably.

Is it very well possible that in the future multiple parties will act as identity provider.

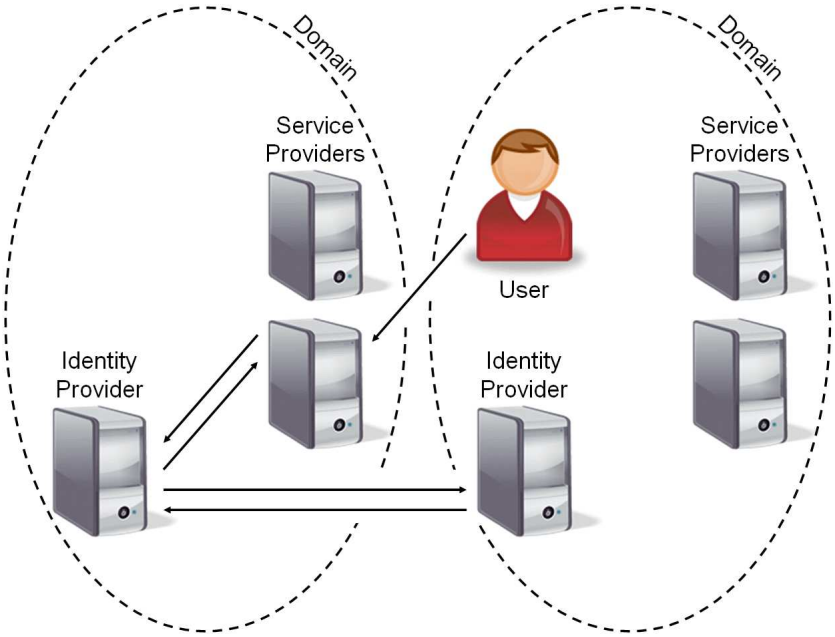

Figure 1.2: *Traditional Federated Identity Management.* A user wants a service from a provider in an other domain. The identity provider in the user's domain is involved to authenticate the user.

1.3 A privacy-friendly view on IM

The growth of digital commerce and exchange of identity information raises some problems. First we mention the privacy problem. In most systems users are expected to hand over a lot of identity information, which will be stored in databases for a long time. Because identity systems are more and more linked to each other by means of federation, it is possible to gather enough identity information of a certain user to make a very detailed profile of him. So there is a strong loss of privacy.

At the same time identity information, especially on the internet, but not limited to it, becomes of huge interest to thieves. Digital transactions can have a high real-world value, and with burglary in only one system a lot of 'identities' can be stolen.

Another problem is the inaccuracy of identity information. As the information is scattered throughout many systems, with an update of the information there is a high probability that not all information on these systems will be correctly synchronised.

For these reasons a new approach to identity management is needed. Instead of asking a lot of identity information from someone in order to determine if he is allowed to perform a certain action or start a transaction, it is better to check only if the user has the right credentials to perform the action or transaction. A credential is an attestation of a certain fact about the holder, issued by a party with an authority or assumed competence to do so. Examples of credentials are a driving license, a diploma, a proof of citizenship, something that represents a certain amount of money, etcetera. To show such credentials it is not necessary to show identifying information. Compare this to buying something in a store: it is not needed to give your name, but it suffices to hand over one or more bills. In other words: instead of storing a lot of data about a user and his credentials on a lot of servers, the user himself stores this data and these credentials, and only reveals them when needed.

Therefore we introduce the concept of *anonymous credential systems*. In such identity systems there are no identifiers of the specific user needed, but only credentials suffice to give a user access to a service. The identity providers do not provide identifying information anymore, but issue credentials instead.

Because it often is useful to keep a relationship between identity provider and the user or between service provider and the user, it is possible to agree on an identifying name for that relation, a *pseudonym*. Therefore anonymous credential systems can also be called *pseudonym systems*.

This privacy-friendly view on identity management preserves privacy for the users and increases the threat of identity theft.

1.4 Laws for successful Identity Management

We have seen that it is better to ask credentials instead of identifying information from users. Whether or not a (federated) identity management system will be widely accepted depends a lot on the trust and convenience experienced by the users and service-providers. In 2005, Seven Laws of Identity were published as result of a project by Kim Cameron [Cam05]. It is expected that these laws have to be used as a starting point of an identity management system to make it successful.

The seven laws are:

1. **User Control and Consent**
   Digital identity systems must only reveal information identifying a user with the user's consent.

2. **Limited Disclosure for Limited Use**
   The solution which discloses the least identifying information and best limits its use is the most stable, long-term solution.

3. **The Law of Fewest Parties**
   Digital identity systems must limit disclosure of identifying information to parties having a necessary and justifiable place in a given identity relationship.

4. **Directed Identity**
   A universal identity metasystem must support both "omnidirectional" identifiers for use by public entities and "unidirectional" identifiers for private entities, thus facilitating discovery while preventing unnecessary release of correlation handles.

5. **Pluralism of Operators and Technologies**
   A universal identity metasystem must channel and enable the interworking of multiple identity technologies run by multiple identity providers.

6. **Human Integration**
   A unifying identity metasystem must define the human user as a component integrated through protected and unambiguous human-machine communications.

7. **Consistent Experience Across Contexts**
   A unifying identity metasystem must provide a simple consistent experience while enabling separation of contexts through multiple operators and technologies.

The next section introduces *idemix*, a pseudonym system which respects the first four laws.

1.5 The pseudonym system *idemix*

idemix is a pseudonym system developed by IBM Research[1]. The research of pseudonym systems and development of *idemix* is done by, amongst others, Jan Camenisch and Anna Lysyanskaya.

[1] `http://www.zurich.ibm.com/security/idemix/`

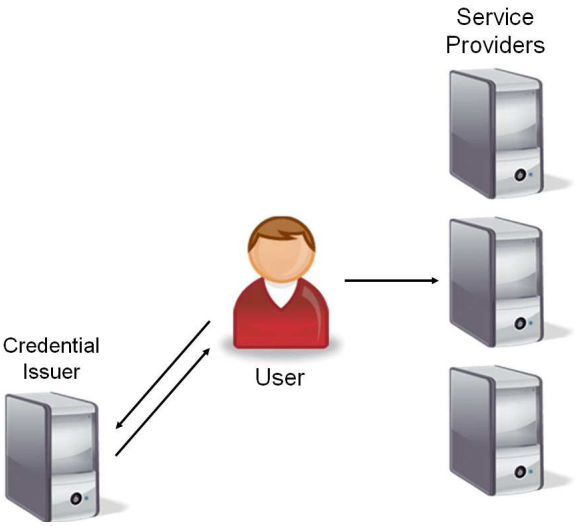

Figure 1.3: *Privacy-friendly Federated Identity Management.* A credential issuer provides a credential to the user. This credential suffices for the user to get service at the service provider.

This system consists of users and organisations. The organisations issue credentials to the users and/or verify if a user owns specific credentials. We will call these organisations issuers, respectively verifiers. For organisations it is possible to be an issuer and a verifier in the same transaction, for example when issuing a credential requiring another credential.

For a user it is possible to use a pseudonym and to obtain a credential from an issuing organisation, and later to show this credential to another organisation while using another pseudonym.

Even if all organisations cooperate, they cannot link a pseudonym under which a credential is verified to the pseudonym under which it was originally issued. It is also possible to show a credential several times to the same organisation, without letting the organisation know that each time this is done by the same user using the same credential.

In *idemix* it is impossible to obtain a credential without asking the right issuer to do so. Counterfeiting therefore is impossible.

When a credential is issued to a user, the credential is bound to a pseudonym. A credential is bound to a certain user, however an issuer or verifier cannot see to which user. It is impossible to pass on a credential that is bound to a pseudonym of a certain user to a pseudonym of a different user.

It is still possible to lend out pseudonyms and credentials to another user, but *idemix* has properties that make this very unattractive.

As we can see, the first four of the Seven Laws are respected by *idemix*:

1. The user is always needed in transactions where his credentials are used, and therefore has control and gives consent to the use identifying information.

2. Pseudonyms and credentials have only to be shown when needed.

3. Only parties that are really needed are involved in a transaction in *idemix*.

4. The user can choose a pseudonym for a specific relation which he does not use elsewhere, so these are unidirectional. Organisations create for themselves public keys which are omnidirectional.

Because *idemix* is an identity system that can be implemented in a meta-system, we cannot apply the fifth law directly on *idemix*. The human integration and experience, which the sixth and seventh law focus on, depend mainly on the implementation.

Therefore, with an appropriate implementation, it is interesting to use the protocols of *idemix* as a basis for an identity management system.

1.6 Using a smart card in IM systems

In February 2007, IBM announced a release of *idemix* as java code in the open source project Higgins[2]. Releasing this software with the source code has several advantages. For example, it is possible to check the code and compile it yourself, so that it is not possible to place code in it that leaks information. Software can easily been distributed over the Internet or in other ways and it can be implemented on very different types of devices, from desktop computers to cell phones or dedicated devices such as digital wallets. However, a software only implementation has some unpleasant drawbacks. The most important of these drawbacks is the problem of the key storage. It is almost impossible to store secret keys in a software environment without the danger of loss, theft, modification or unauthorised use. Also keys can be easily copied and shared with others.

A solution to this is to use a smart card for some parts of the implementation of *idemix*. Hopefully we can keep the advantages of the software implementation and use the smart card to solve some problems of the software only implementation.

[2] `http://www.eclipse.org/higgins/`

Figure 1.4: A smart card.

We mention some of the advantages:

- A smart card can be made tamper proof, at least a lot more than software can be made. So we can store a masterkey in it which can not be read out or modified and will be made inoperative if someone tries to modify or read out.
- A smart card can contain a printed name, age, gender and photo and thus can be bound to a real person.
- A smart card can be centrally distributed by identity providers.
- A smart card can be easily protected against viruses and Trojan horses.
- A smart card can require the user to enter a PIN.
- A smart card is portable.

These advantages and even more are also mentioned by Stefan Brands in his PhD thesis [Bra00].

Moreover, governments develop electronic identity cards, in most cases as a smart card. *idemix* in combination with these electronic identity smart cards could be used to interact online with governmental organisations. Other organisations could use *idemix* in combination with these cards to assure that there are real persons behind the pseudonyms they know.

This combination of smart cards and *idemix* would lead to a very secure and privacy-friendly system for identity management.

1.7 Implementing a smart card in *idemix*

The previous sections introduce the pseudonym system *idemix* and explain that using a smart card in an identity management system has obvious advantages. But because *idemix* uses a lot of heavy mathematical operations and a smartcard has limitations on its calculation power and storage possibilities, the question remains if it is possible to use smart cards with *idemix*. The research I have done at TNO ICT, and what is written in the remaining part of this thesis, results from the following research question:

In which way can *idemix* be implemented in a system which makes use of

1. **an electronic identity card (smart card) with limited computation power and limited memory,**
2. **a terminal or a system reachable via a terminal with relatively much computing power and memory,**

and where at the same time all necessary secret keys of the citizen are contained on and handled by the electronic identity card?

1.8 Outline of this thesis

The next chapter gives some preliminaries we need in order to understand *idemix*. In Chapter 3 the properties of *idemix* and the *idemix* protocol are introduced and also the building block of *idemix* are explained there. For *idemix* we have to use *zero-knowledge proofs of knowledge*, which are explained in the subsequent chapter. In Chapter 5 we investigate if it is possible to use a smart card and terminal in *idemix*, as described in the research question. The conclusions with respect to the research question and some suggestions for further research are given in the final chapter.

Information about calculating square roots of large numbers, modular arithmetic on smart cards and state-of-the-art smart card specifications can be found in the Appendices.

1.9 TNO exploring Federated Authentication and Identity Management

This master's thesis is a result of an internship at TNO ICT in Groningen.

The Netherlands Organisation of Applied Scientific Research (de Nederlandse Organisatie voor Toegepast Natuurwetenschappelijk Onderzoek) is an independent and non-profit research institute which focuses on the practical application of scientific knowledge.

Since TNO was established by law in 1930, it has supported companies, governments and public organisations with innovative and practicable knowledge. Nowadays TNO has approximately 5000 employees who carry out contract research and provide specialist consultancy. They also audit and certify products and services, based on independent and scientifically founded judgements. About 30 percent of the projects are not on contract basis, but are internal knowledge acquiring projects. This is done in order to keep the knowledge of the employees up-to-date.

My research is done as a part of the FAIM (Federated Authentication and Identity Management) project, an internal project of the ICT Security group to acquire knowledge about identity management and federation. By keeping this knowledge up-to-date, TNO is able to give the best advice about identity management and federation to the Dutch government and companies like the Dutch telecom provider KPN.

Chapter 2

Preliminaries

If we want to understand how *idemix* (Chapter 3) and zero-knowledge proofs (Chapter 4) work, we first have to introduce some notation, definitions, assumptions and lemmas. This introduction is done in this chapter.

2.1 Notation

We introduce some notation. It is important to mention that in *idemix* all values are represented by bitstrings. Interpreting these as binary numbers, all considered values are positive integers. When we use the logarithm log, we mean the binary log, thus log_2. When choosing random values, denoted by $\in_R$, we also mean integers. Also for intervals, using the notation $[a, b]$ and (a, b), we mean integer intervals, including, respectively, excluding the endpoints.

2.2 Definitions

Definition 2.2.1 (Special RSA modulus)**.** *A* RSA modulus $n{=}pq$ *is* special *if p and q are distinct primes of the form $p = 2p' + 1$ and $q = 2q' + 1$ where p', q' are also prime numbers.*

Definition 2.2.2 (Integers modulo n)**.** *The* integers modulo n*, denoted by $\mathbb{Z}/n\mathbb{Z}$ or by $\mathbb{Z}_n$, is the set of (equivalence classes of) integers $\{0, 1, 2, \ldots, n-1\}$, where m corresponds to the class $m + n\mathbb{Z} \subset \mathbb{Z}$. If $n = p$ a prime, then $\mathbb{Z}_p$ is a field.*

Definition 2.2.3 (The unit group $\mathbb{Z}_n^*$)**.** *$\mathbb{Z}_n^*$ is the* group of units of the ring *$\mathbb{Z}_n$. To be precise, these are the $m \in \mathbb{Z}_n$ such that* $\gcd(m, n) = 1$.

Definition 2.2.4 (Euler totient function)**.** *The* Euler totient function $\phi(n)$ *is the number of positive integers less than or equal to* n *that are coprime to* n*.*

Note: $\phi(n)$ gives the order of $\mathbb{Z}_n^*$. If $n = pq = (2p' + 1)(2q' + 1)$ is a special RSA modulus, then $\phi(n) = (p - 1)(q - 1) = 4p'q'$.

Definition 2.2.5 (The group of Quadratic Residues QR_n)**.** *An integer* $d \in \mathbb{Z}_n^*$ *is a* quadratic residue modulo n*, if there exists an integer* c *such that* $c^2 \equiv d \bmod n$*. All integers satisfying this requirement form the* group of Quadratic Residues $QR_n \subset \mathbb{Z}_n^*$.

For an extensive introduction see for example chapter 2 of [MOV96], chapter 11 of [Sch96] or other books and sheets on Number Theory and Cryptology.

Indistinguishability

The definitions in this section are derived from a paper by Damgård and Nielsen [DN07].

Consider a probabilistic algorithm U. For every possible string y there is a probability $U_x(y)$ that y is output when x was the input. We define U_x as the probability distribution of U's output on input x.

Now we define statistical distance:

Definition 2.2.6 (Statistical distance)**.** *Given two probability distributions* P, Q*, the* statistical distance *between them is defined to be* $SD(P, Q) = \sum_y |P(y) - Q(y)|$*, where* $P(y)$ *(or* $Q(y)$*) is the probability* P *(or* Q*) assigns to* y*.*

Using this, we have

Definition 2.2.7 (Indistinguishability)**.** *Given two probabilistic algorithms (or families of distributions)* U, V*, we say that*

- U, V *are* perfectly indistinguishable*, if* $U_x = V_x$ *for every* x*.*
- U, V *are* statistically indistinguishable*, if* $SD(U_x, V_x) \leq 1/p(|x|)$ *for every positive polynomial* p *and every string* x *of sufficiently large length* $|x|$*.*

- U, V *are* computationally indistinguishable*, if the following holds for every probabilistic polynomial time algorithm D:*
  let $p_{U,D}(x)$ be the probability that D outputs "U" as its guess, when D's input comes from U, and similarly $p_{V,D}(x)$ be the probability that D outputs "U" as its guess, when D's input comes from V. Then $|p_{U,D}(x) - p_{V,D}(x)| \leq 1/p(|x|)$ for every positive polynomial p and every sufficiently large string x.

Loosely speaking, two families of distributions are statistically indistinguishable if they are "statistically" so close to each other that nobody can tell them apart. Two families of distributions can be statistically different, but if it it unfeasible to detect this in a reasonable amount of time, we call them computationally indistinguishable. Note that statistical indistinguishability implies computational indistinguishability.

2.3 Assumptions

The security of *idemix* is based on the strong RSA assumption and the decisional Diffie-Hellman assumption.

Assumption 2.3.1 (Strong RSA assumption (SRSA assumption))**.** *The following problem, i.e., the* flexible RSA problem*, is hard to solve: Given a RSA modulus n of unknown factorisation and a random $z \in \mathbb{Z}_n^*$, find $r > 1$ and $y \in \mathbb{Z}_n^*$ such that $y^r = z$.*

Assumption 2.3.2 (Decisional Diffie-Hellman assumption (DDH assumption))**.** *Let G be a cyclic group of order q, with generator g. Choose $a, b, c \in_R \mathbb{Z}_q$ independently. Then the two probability distributions*

$$\langle g^a, g^b, g^{ab} \rangle$$

and

$$\langle g^a, g^b, g^c \rangle$$

are computationally indistinguishable.

2.4 Lemmas

The following lemmas are also found as Lemma 4.1.5, Corollary 4.1.8 and Lemma 4.1.10 in Lysyanskaya's PhD thesis [Lys02].

Lemma 2.4.1. *Let a composite integer n be given. Given any value x such that $\phi(n)|x$, one can find a non-trivial divisor of n in probabilistic polynomial time.*

Proof. We distinguish three different cases:

1. n is an even number.
2. n is an odd number and is of the form $n = p^e$, with p an odd prime and $e \geq 2$.
3. n is an odd number and is of the form $n = p \cdot q \cdot r$, with p, q odd primes and r a rest.

In the first case, 2 is a non-trivial factor and so we are finished.

Now suppose n is odd and of the form $n = p^e$. Then we can check whether $\sqrt[k]{n}$ is an integer for all possible values of k. Because $\sqrt[e]{n} \geq 3$, we know that $e \leq \frac{\log n}{\log 3}$. Thus we only have to test for $1 < k \leq \left\lfloor \frac{\log n}{\log 3} \right\rfloor$.

The third case is that n consists of at least two different prime factors, which we call p and q. We remove all even factors from $p-1$ and $q-1$ by dividing by 2^j with j as large as possible. The results are denoted by p' and q'. So let $p' = (p-1)/2^{j_p}$, $q' = (q-1)/2^{j_q}$ be odd integers. Assume without loss of generality that $j_p \leq j_q$.

Now we can write $x = y \cdot \phi(n) = y \cdot (p-1) \cdot (q-1) \cdot \tilde{r} = y \cdot p' \cdot q' \cdot \tilde{r} \cdot 2^{j_p + j_q}$, for some factor $\tilde{r}$ of $\phi(n)$. Define and calculate x' by removing the factors 2 from x. Thus $x' = p' \cdot q' \cdot r'$ for some odd $r' \geq 1$.

Now select a random $u \in \{1, \ldots, n-1\}$. If $u \notin \mathbb{Z}_n^*$, then $\gcd(u, n)$ is a non-trivial factor of n. So assume $u \in \mathbb{Z}_n^*$ and define $V := u^{2^{j_p - 1} \cdot x'}$. (Note that at this point we do not know the value of j_p.)

With probability $1/2$, u is a square modulo p, and so

$$V = u^{2^{j_p-1} \cdot x'} = (v^2)^{2^{j_p-1} \cdot x'} = v^{2^{j_p} \cdot p' \cdot q' \cdot r'} = (v^{q'r'})^{p-1} \equiv 1 \bmod p$$

With independent and non-negligible probability, u is a generator modulo q. Because $j_p \leq j_q$, it holds that $q - 1 = 2^{j_q} \cdot q' \nmid 2^{j_p - 1} \cdot x'$ and so $u^{2^{j_p-1} \cdot x'} \neq 1 \bmod q$.

Therefore, if j_p were known, we would get with non-negligible probability a value $V = u^{2^{j_p-1} \cdot x'} \bmod n$ such that $V \equiv 1 \bmod p$ and $V \neq 1 \bmod q$. Thus $p \leq \gcd(V - 1, n) < n$, and so $\gcd(V - 1, n)$ is a non-trivial divisor of n.

Because there are only $\log n$ possibilities for j_p, we can try them all.

So our algorithm is as follows:
for random $u \in \{0, \ldots, n-1\}$, we calculate

$$\begin{aligned} V_0 &= u^{x'} \\ V_1 &= u^{2x'} \\ V_2 &= u^{2^2 x'} \\ &\vdots \\ V_i &= u^{2^i x'}, \end{aligned}$$

and compute $\gcd(V_i - 1, n)$.

We do this for $0 \leq i \leq \log n$, and hopefully we have a $\gcd(V_i - 1, n)$ which gives a factor of n. Otherwise we try again with another random u. □

Lemma 2.4.2. *Given a special RSA modulus n, and an integer x such that*

$$\gcd(\phi(n), x) > 4,$$

one can efficiently factor n.

Proof. We know that $\phi(n) = 4p'q'$, thus $\gcd(\phi(n), x) > 4$ gives us that x contains p', q' or both.

Suppose x contains $p'q'$, then $4x$ is a multiple of $\phi(n)$, so Lemma 2.4.1 gives us the factors p and q.

Otherwise suppose, without loss of generality, that x contains p', but does not contain q'. Then choose $u \in_R \mathbb{Z}_n$, define $V := u^{2x}$ and by the same reasons as in Lemma 2.4.1, $V \equiv 1 \bmod p$, $V \neq 1 \bmod q$ with non-negligible probability. Therefore $p \leq \gcd(V - 1, n) < n$, and so $\gcd(V - 1, n)$ is a non-trivial divisor of n. □

Lemma 2.4.3. *Let a special RSA modulus $n = pq$, $p = 2p' + 1$, $q = 2q' + 1$, be given. Suppose we are given the values $u, v \in QR_n$ and $x, y \in \mathbb{Z}$, $x \nmid y$ such that $v^x \equiv u^y \bmod n$. Then, values $z, w > 1$ such that $z^w \equiv u \bmod n$ can be computed efficiently.*

Proof. Let $c = \gcd(x, y)$. If $\gcd(4c, \phi(n)) > 4$, then by Lemma 2.4.2, we factor n. Otherwise, because $\phi(n) = 4p'q'$, it must be the case that $\gcd(c, p'q') = 1$. Therefore, there exists a value $d \in \mathbb{Z}^*_{p'q'}$ such that $cd \equiv 1 \bmod p'q'$.

Note that

$$v^{x/c} \equiv (v^x)^d \equiv (u^y)^d \equiv u^{y/c} \bmod n.$$

By the Extended GCD algorithm (see for example [MOV96]), find integers a and b such that $a(x/c) + b(y/c) = \gcd(x/c, y/c) = 1$.
Let $z = u^a v^b \bmod n$. Then

$$z^{x/c} \equiv u^{a(x/c)} v^{b(x/c)} \equiv u^{a(x/c)} (v^x)^{b/c} \equiv u^{a(x/c)} (u^y)^{b/c} = u^{a(x/c)+b(y/c)} = u \bmod n.$$

From $x \nmid y$ it follows that $c = \gcd(x, y) < x$.
Thus take the integer $w = x/c > 1$ to obtain

$$z^w = z^{x/c} \equiv u \bmod n.$$

□

Chapter 3

The pseudonym system *idemix*

In this chapter we describe how *idemix* works. First we list the required and desired properties, and after that we look at how these requirements are mathematically achieved.

In 1985, Chaum introduced the concept of anonymous credential systems (also called pseudonym systems) [Cha85]. An anonymous credential system consists of users and organisations. Issuance and verification of credentials is done by the organisations, while users preserve their anonymity. In brief there are two requirements for an anonymous credential system, namely security for the credential issuers and verifiers and privacy for the users. Security for the issuers and verifiers means that credentials can not be forged by users or other organisations, even if they work together or launch an adaptive attack[1]. Only users that really own credentials can prove ownership to the verifiers. If users are not allowed to have certain credentials, it must also be impossible to obtain them by cooperation.

Security can also mean that credentials cannot be shared. This can never be really guaranteed, because users can give away all there secrets. However, it can be made very unattractive for users to do so.

At the same time users must stay anonymous (or pseudonymous). That is, organisations can not find out more information about the user than that he owns certain credentials. Moreover, pseudonyms and credentials belonging to a user cannot be linked.

[1] In an adaptive attack the next step of the attack is adapted accordingly the results of earlier steps in the attack

idemix is such a pseudonym system, as the basic requirements are fulfilled by three system properties:

1. **Unlinkable pseudonimity**
   Users use pseudonyms to interact with organisations. Organisations cannot obtain information about the identity of the users from this pseudonym. Organisations can neither link two different pseudonyms to each other, nor link two different uses of a credential bound to a certain pseudonym. At the same time the pseudonyms are bound to a user, so users cannot exchange credentials without exposing secret identity information.

2. **Unforgeable pseudonymous credential granting**
   Using a signature scheme, an organisation can grant credentials by signing the combination of a pseudonym and a value representing the credential. In order to sign a credential, the organisation needs to know only the pseudonym, thus the master key and real identity of the user remain secret to the organisation.

3. **Zero-knowledge credential verification**
   By using combinations of zero-knowledge proofs of knowledge, the user can show to an organisation ownership of a credential without revealing his identity or pseudonym with the issuer.

In order to make the formulas more readable, we use the following letters in the subscript of the variables: U stands for User, O for Organisation, I for Issuer and V for Verifier. The combination UO means that it is a variable of the User to use with a certain Organisation.

3.1 Unlinkable pseudonimity

A pseudonym in *idemix* has two requirements:

1. The pseudonym has to be **binding** to the user, so that once the pseudonym is established, the user bound to it cannot be changed.

2. The pseudonym has to be **hiding** the user's identity, so that the organisation can not link the pseudonym to the real identity or to other pseudonyms of the same user.

To establish a pseudonym in *idemix*, we use a commitment scheme to commit to a value hidden to the verifier. Also for executing the zero-knowledge proofs we need a commitment scheme.

A commitment scheme has two requirements:

1. The commitment has to be **binding** to the value for the prover. If asked to reveal a value that fits the commitment, the prover can not reveal another value than the chosen one.

2. The commitment has to be **hiding** the value for the verifier. This is defined as: in the view of the verifier all possible chosen values are equally likely put into the commitment.

Pedersen [Ped92] proved that a commitment of the form

$$X = g^x h^y \bmod p,$$

where p is a prime number, has the required properties. Here X is the commitment to x, while y is used as a hiding exponent. The values g and h are public bases, and also p is public.

Based on this, in *idemix* the pseudonym is of the form

$$P_{UO} = a_O^{x_U} b_O^{s_{UO}} \bmod n_O.$$

Because the signature scheme used by *idemix* requires n_O to be a special RSA modulus, our group has a hidden order. That gives some requirements on a_O and b_O, as will be mentioned in section 4.3.

3.2 Unforgeable pseudonymous credential granting

Credential issuers grant credentials to users by signing pairs that consists of a pseudonym and a value representing the credential. By signing a pseudonym that hides the user's master key, the anonimity of the user is preserved. The choice of a signature scheme makes it impossible to forge these credentials. In her PhD thesis [Lys02], Lysyanskaya gives a signature scheme that satisfies the requirements:

Set-up. Choose a special RSA modulus $n_O = p_O q_O$, $p_O = 2p'_O + 1$, $q_O = 2q'_O + 1$, with p_O, q_O, p'_O, q'_O primes and n_O of length ℓ_n.
Choose $a_O, b_O, d_O \in QR_{n_O}$ uniformly at random. Output as public key $PK_O = (n_O, a_O, b_O, d_O)$ and keep as secret key $SK_O = (p_O, q_O)$.

Signing. To sign a pseudonym $P_{UO} = a_O^{x_U} b_O^{s_{UO}}$, which hides the user's master key $x_U \in [0, 2^{\ell_x})$ and a credential value, we choose a random prime $2^{\ell_e - 1} < e < 2^{\ell_e}$ of length $\ell_e = \ell_x + 2$ and a random number r of length $\ell_s = \ell_n + \ell_x + \ell$, with ℓ a security parameter.

We create a signature on P_{UO} and d_O by calculating

$$c = (P_{UO}\, b_O^r\, d_O)^d \bmod n_O,$$

with d such that $ed = 1 \bmod \phi(n_O)$.
Output as a signature (c, e, r).

Verification. To verify a signature on a message,
check that $c^e = a_O^{x_U} b_O^{s_{UO}+r} d_O \bmod n_O$ and that $2^{\ell_e - 1} < e < 2^{\ell_e}$.

Theorem 3.2.1. *This signature scheme is secure (=unforgeable) under the Strong RSA assumption.*

Proof. A proof is given in detail in a paper of Camenisch and Lysyanskaya [CL02], and in Lysyankaya's PhD thesis [Lys02]. □

We see that a credential is a triple (c, e, r) that satisfies the following equation:

$$c^e = P_{UO}\, b_O^r\, d_O \bmod n_O.$$

3.3 Zero-knowledge credential verification

In *idemix* a user can prove ownership of a credential, and also that both the credential and the pseudonym by which the user is known to the verifier are bound to the same master key.

This is done by using combinations of zero-knowledge proofs of knowledge. These proofs prove only the knowledge of credential and pseudonym values, without giving away information about the actual values. More detailed information about these proofs can be found in the next chapter. It is shown there that verification always succeeds if the prover has knowledge of the values, but if he does not know the values, he only succeeds with negligible probability.

3.3.1 Notation of zero-knowledge proofs of knowledge

Camenisch and Stadler [CS97] have introduced a notation for zero-knowledge proofs of knowledge. This notation makes clear which knowledge is proven, while hiding the technical details of the underlying mathematics. Because the combinations of basic zero-knowledge proof of knowledge in *idemix* are very complex, this notation helps us to clarify the protocols given in the next sections.

The notation is explained by an example: a proof of knowledge, denoted by

$$\mathrm{PK}\left\{ (\alpha, \beta) : y = g^{\alpha} \bmod n \wedge z = g^{\beta} h^{\alpha} \bmod n \wedge \alpha \in A \right\},$$

means a zero-knowledge proof of knowledge of of the discrete logarithm of y to the base g and of a representation of z to the bases g and h, and additionally the discrete logarithm of the h-part of the representation has to be equal to the discrete logarithm of y to the base g and has to lie in the interval A. This is equal of proving knowledge of the values of the greek letters in the equations on the right side of the colon. Throughout this document, we use small greek letters for the elements whose knowledge has to be proven and all other letters denote elements that are known to both the prover and the verifier.

To preserve clarity of the notation, the modulus will not be mentioned when it follows directly from the context. Whenever we mention a *proof of knowledge* in this thesis, we mean a *zero-knowledge proof of knowledge*.

3.4 Basic actions

The basic version of *idemix* consists of five actions. SetUp lets a user or organisation enter the system. FormNym lets a user and organisation generate a pseudonym for the user. With GrantCred, an organisation can grant a credential to a user. Ownership of a credential can be verified with VerifyCred. And by executing VerifyCredOnNym it can be verified that the user owns a credential that is bound to the user which is hidden in a certain pseudonym.

To obtain the non-basic properties of *idemix*, a number of additional actions and modifications of the basic ones are used. These are not discussed in this thesis; we refer to [CL01a] for details on them.

For each action we will do an analysis on its time complexity. As we can see in Appendix B, calculation complexity mainly depends on the number of exponentiations. We use this number here as a unit of time. How much time an exponentiation costs on a state-of-the-art smart card can be found in the afore mentioned appendix.

A zero-knowledge proof of knowledge also involves exponentiations. For each exponent the user wants to prove knowledge of, he has to do an exponentiation. We will explain this in Chapter 4.

Interval proofs also need exponentiations. However, the number of exponentiations needed here depends on properties of the interval and the precise form of the interval proof. So here we only count the number of interval proofs.

3.4.1 System initialisation

When initializing the system, a central party has to choose appropriate values for the parameters ℓ_n, ℓ_x, ℓ_e and ℓ_s. Here ℓ_n denotes the bitlength of n (etc.) The security of the system depends on the choice of these parameters. See section 3.5 for some possible choices.

3.4.2 Entering the system

When entering the system, an organisation has to choose a public and a secret key. If a user enters the system, he must choose a master key.

SetUp(O) *Setup phase for an organisation*

1. O chooses primes p'_O and $q'_O \in_R [2^{\ell_n/2-2}, 2^{\ell_n/2-1})$, such that $p_O = 2p'_O+1$ and $q_O = 2q'_O + 1$ are prime. O sets modulus $n_O = p_O q_O$. Now n_O is a ℓ_n-bit special RSA modulus with large factors.

2. O chooses random $a_O, b_O, d_O, g_O, h_O \in QR_{n_O}$.

3. O stores $SK_O = (p_O, q_O)$ as a secret key and publishes $PK_O = (n_O, a_O, b_O, d_O, g_O, h_O)$ as his public key.

SetUp(U) *Setup phase for a user*
The user chooses a master key $x_U \in_R [0, 2^{\ell_x})$.

Security Remark

Note that for security of the signature scheme, the values a_O, b_O, d_O, g_O and h_O have to be elements in the group of quadratic residues modulo n_O, i.e.,

in QR_{n_O}. This has to be proven in each function and can be done by an extra zero-knowledge proof of knowledge. However, this can be avoided by executing the proof of knowledge with squared bases. For example, executing $\mathrm{PK}\left\{(\alpha) : y^2 = (g^2)^\alpha\right\}$ instead of $\mathrm{PK}\left\{(\alpha) : y = g^\alpha\right\}$.

Analysis

Finding the safe primes p_O and q_O can take some time on a standard desktop pc. A probabilistic primality test for values of this size take for example about a minute. More robust primality tests take a multiple of this, but it can be done within an hour. Because this only has to be done at the setup phase, it does not influence the speed of the system.

3.4.3 Generating a pseudonym

In order to obtain a credential, a pseudonym by which the user is known to the organisation must be established first.

FormNym(U, O)
At the start of this protocol the user has his master key x_U and both parties have the public key of the organisation. After execution a pseudonym $P_{UO} = a_O^{x_U} b_O^{s_{UO}}$ will be established. Both parties store P_{UO}. The values x_U and s_{UO} are only known by the user.

1. U chooses $s_{UO} \in [0, 2^{\ell_n})$ and sets $P_{UO} = a_O^{x_U} b_O^{s_{UO}} \bmod n_O$ and sends P_{UO} to O.

2. U proves knowledge of the master key and the blinding exponent to O. He also proves that the master key is chosen from the correct interval. This is done by executing

$$\mathrm{PK}\left\{(\alpha, \beta) : P_{UO}^2 = (a_O^2)^\alpha (b_O^2)^\beta \wedge \alpha \in [0, 2^{\ell_x}) \wedge \beta \in [0, 2^{\ell_n})\right\}.$$

3. Both parties store P_{UO} and the user U also stores s_{UO}.

Analysis

By setting the pseudonym the user does a ℓ_x-bit and a ℓ_n-bit modular exponentiation. For the proof of knowledge the user also has to do 2 modular exponentiations. Moreover, 2 interval proofs are needed.
Total: 4 exponentiations, 2 interval proofs.

3.4.4 Granting a credential

A credential is granted by signing a pseudonym in combination with a value representing the credential.

$\mathsf{GrantCred}(\mathsf{P_{UO}}, \mathsf{O})$
At the start of this protocol the organisation knows the user under a pseudonym. After execution the user has gotten a triple (c_{UO}, e_{UO}, r_{UO}) as a signature on the tuple (P_{UO}, d_O).

1. U sends P_{UO} to O and proves knowledge of the master key (and thus ownership of the pseudonym) by executing

$$\mathrm{PK}\left\{ (\alpha, \beta) : P_{UO}^2 = (a_O^2)^\alpha (b_O^2)^\beta \right\}.$$

2. O chooses a random prime $e_{UO} \in [2^{\ell_e - 1}, 2^{\ell_e})$ and a random $r_{UO} \in [0, 2^{\ell_s})$ and computes

$$c_{UO} = (P_{UO}\, b_O^{r_{UO}} d_O)^{1/e_{UO}} \bmod n_O.$$

   O sends the triple (c_{UO}, e_{UO}, r_{UO}) to U.

3. U verifies that $c_{UO}^{e_{UO}} = P_{UO}\, b_O^{r_{UO}} d_O \bmod n_O$ and that $e_{UO} \in [2^{\ell_e - 1}, 2^{\ell_e})$.

4. U and O store the triple.

Analysis

The user has to prove knowledge of the exponents in the pseudonyms and therefore has to do 2 modular exponentiations. For verification of the credential triple, the user must calculate 2 exponentiations.
Total: 4 exponentiations.

3.4.5 Verifying a credential

Verification of ownership of a credential is done by verifying that the user knows all necessary values.

$\mathsf{VerifyCred}(\mathsf{O}, \mathsf{P_{UO}})$
At the start of this protocol the user and the verifying organisation know the public key of the issuing organisation, the user also has a tuple (x_U,s_{UO}) which defines the pseudonym P_{UO} and a triple (c_{UO}, e_{UO}, r_{UO}) as a signature on the pseudonym and the credential.

1. U chooses $r_1, r_2 \in [0, 2^{2\ell_n})$, computes $A = c_{UO} h_O^{r_1} \bmod n_O$ and $B = g_O^{r_2} h_O^{r_1} \bmod n_O$ and sends the results to V.

2. U proves knowledge of the exponents in the pseudonym and knowledge of a signature by executing

$$\begin{aligned} \mathrm{PK}\Big\{ (\alpha,\beta,\gamma,\delta,\epsilon,\zeta,\xi) : d_O^2 = (A^2)^\alpha (\frac{1}{a_O^2})^\beta (\frac{1}{b_O^2})^\gamma (\frac{1}{h_O^2})^\delta \wedge \\ B^2 = (h_O^2)^\epsilon (g_O^2)^\zeta \wedge 1 = (B^2)^\alpha (\frac{1}{h_O^2})^\delta (\frac{1}{g_O^2})^\xi \wedge \\ \beta \in [0, 2^{\ell_x}) \wedge \gamma \in [0, 2^{\ell_n}) \wedge \alpha \in [2^{\ell_e - 1}, 2^{\ell_e}) \Big\} . \end{aligned}$$

Explanation of the proof of knowledge

The user has to prove ownership of a credential triple (c_{UO}, e_{UO}, r_{UO}), such that $c_{UO}^{e_{UO}} = a_O^{x_U} b_O^{s_{UO}+r_{UO}} d_O \bmod n_O$. Because we can only prove knowledge of exponents with our zero-knowledge proofs of knowledge and do not want to give away information about c_{UO}, we first have to make a new hiding commitment to the value of c_{UO}. This is done by calculating and sending $A = c_{UO}\, h_O^{r_1} \bmod n_O$. Then we can prove that we know the exponents in

$$A^{e_{UO}} = c_{UO}^{e_{UO}} h_O^{r_1 \cdot e_{UO}} = a_O^{x_U} b_O^{s_{UO}+r_{UO}} d_O h_O^z \bmod n_O, \quad z = r_1 \cdot e_{UO},$$

and that x_U and e_{UO} lie in the right intervals. All this can be done by the following proof of knowledge:

$$\begin{aligned} \mathrm{PK}\Big\{ (\alpha,\beta,\gamma,\delta,\epsilon,\zeta,\xi) : d_O^2 = (A^2)^\alpha (\frac{1}{a_O^2})^\beta (\frac{1}{b_O^2})^\gamma (\frac{1}{h_O^2})^\delta \wedge \\ \beta \in [0, 2^{\ell_x}) \wedge \gamma \in [0, 2^{\ell_n}) \wedge \alpha \in [2^{\ell_e - 1}, 2^{\ell_e}) \Big\} . \end{aligned} \qquad (*)$$

To guarantee soundness of the proof above, we also have to check that the power of h_O is constructed in the right way, i.e., $z = r_1 \cdot e_{UO}$. Therefore we have to make a commitment to r_1 in advance, by calculating and sending $B = g_O^{r_2} h_O^{r_1}$. To prove the right construction of z, we prove knowledge of r_1 by the zero-knowledge proof of knowledge

$$\mathrm{PK}\Big\{ (\epsilon,\zeta) : B^2 = (h_O^2)^\epsilon (g_O^2)^\zeta \Big\} ,$$

and we prove the right construction by proving that $B^{e_{UO}} = h_O^{r_1 \cdot e_{UO}} g_O^{r_2 \cdot e_{UO}} = h_O^z g_O^{r_2 \cdot e_{UO}}$, which can be done by executing

$$\mathrm{PK}\Big\{ (\alpha,\delta,\xi) : 1 = (B^2)^\alpha (\frac{1}{h_O^2})^\delta (\frac{1}{g_O^2})^\xi \Big\} ,$$

where α and δ has to be proven to be equal to the α, resp. δ, in the first proof $(*)$.

Analysis

In the first step, the user has to do 2 times a $2\ell_n$-bit exponentiation. For proving knowledge of the exponents in the signature and knowledge of the signature, the user must exponentiate 8 times and do 3 interval proofs.
Total: 10 exponentiations, 3 interval proofs.

3.4.6 Verifying a credential on a pseudonym

To preserve unlinkability, the pseudonym to which a credential is granted will not be shown to the verifier. However, if the user is known to the verifier by a (different) pseudonym, the user can prove that this pseudonym and the credential are bound to the same master key. This is done by an extension of VerifyCred.

VerifyCredOnNym$(\mathsf{I}, \mathsf{P_{UI}}, \mathsf{P_{UV}}, \mathsf{V})$
At the start of this protocol the user and the verifying organisation know the public key of the issuing organisation and a pseudonym by which the user is known to the verifying organisation. The user also has a master key x_U, values s_{UI} and s_{UV} which define the pseudonyms P_{UI} and P_{UV} and a triple (c_{UI}, e_{UI}, r_{UI}) as a signature on the pseudonym and credential.

1. U chooses $r_1, r_2 \in [0, 2^{2\ell_n})$, computes $A = c_{UI} h_I^{r_1} \bmod n_I$ and $B = g_I^{r_2} h_I^{r_1} \bmod n_I$ and sends the results to V.

2. U proves knowledge of the exponents in the pseudonym and knowledge of a signature by executing

$$\begin{aligned} \mathrm{PK}\Big\{ (\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \xi, \eta) : d_I^2 = (A^2)^{\alpha} (\frac{1}{a_I^2})^{\beta} (\frac{1}{b_I^2})^{\gamma} (\frac{1}{h_I^2})^{\delta} \wedge \\ B^2 = (h_I^2)^{\epsilon} (g_I^2)^{\zeta} \wedge 1 = (B^2)^{\alpha} (\frac{1}{h_I^2})^{\delta} (\frac{1}{g_I^2})^{\xi} \wedge \\ P_{UV}^2 = (a_V^2)^{\beta} (b_V^2)^{\eta} \bmod n_V \wedge \\ \beta \in [0, 2^{\ell_x}) \wedge \gamma \in [0, 2^{\ell_n}) \wedge \alpha \in [2^{\ell_e - 1}, 2^{\ell_e}) \Big\} . \end{aligned}$$

Analysis

In the first step, the user has to do 2 times a $2\ell_n$-bit exponentiation. For proving knowledge of the exponents in the signature and knowledge of the signature, the user must exponentiate 10 times and do 3 interval proofs.
Total: 12 exponentiations, 3 interval proofs.

3.5 System parameters

Having defined the basic actions in *idemix*, we now discuss the common system parameters.

The system is based on RSA-signatures, so its security depends on properties of the modulus n. This should be a special RSA-modulus of length ℓ_n. The value $\ell_n = 2048$ is considered secure till the year 2020. (The value $\ell_n = 1024$ is expected to become insecure this year.)

The set from which a user's master key x_u is chosen, has to be large enough to make the probability of collisions negligible. Therefore we choose its bitlength $\ell_x = 160$.

For secure signing (as described in section 3.2), we also have to calculate $\ell_e = \ell_x + 2$ and $\ell_s = \ell_n + \ell_x + \ell$, where ℓ is a security parameter. The value of ℓ is chosen such that the simulated distribution in the proof of security is statistically close to the actual distribution. So we choose $\ell = 160$. For details on this, we refer to [Lys02]. It follows that $\ell_e = 162$ and $\ell_s = 2048 + 160 + 160 = 2368$.

Chapter 4

Zero-knowledge proofs

The previous chapter showed how zero-knowledge proofs are used by *idemix* to generate pseudonyms and issue and verify credentials. Here we discuss how these proofs are implemented.

4.1 Zero-knowledge proof

A zero-knowledge proof is a special case of interactive proof systems.

A definition of interactive proof systems is given by [Gol04]:

Definition 4.1.1. *An* interactive proof system for a set S *is a two-party game, between a* verifier *executing a probabilistic polynomial-time strategy* (denoted V) *and a* prover *which executes a computationally unbounded strategy* (denoted P)*, satisfying*

Completeness *For every $x \in S$ the verifier V always accepts after interacting with the prover P on common input x.*

Soundness *For some positive polynomial p, it holds that for every $x \notin S$ and every potential strategy P^*, the verifier V rejects with probability at least $1 - (1/p(|x|))$, after interacting with P^* on common input x.*

We deduce the definition of zero-knowledge interactive proof systems from the definitions by [Gol04] and [DN07]:

Definition 4.1.2. *An interactive proof system for a set S is* zero-knowledge *if, for input $x \in S$ and arbitrary auxiliary input δ (of length at most some fixed polynomial in the length of x), it satisfies:*

Zero-knowledge *For every probabilistic polynomial time verifier V^*, there is a simulator M_{V^*} running in expected probabilistic polynomial time which is indistinguishable from the real interactive proof system.*

A more informal definition of a zero-knowledge proof can be given as follows: If a statement holds, the prover can convince the verifier of it without giving away more information. And if the statement does not hold, the prover succeeds only with a probability smaller than 1.

Of course this probability should be chosen to be negligible in practice. This can be achieved by executing the protocol sufficient many times.

To prove the soundness of a protocol, we have to show that there exists a *knowledge extractor*. Such an extractor has oracle access to the possibly malicious prover (ie. the extractor gets the response of the prover to particular messages it may recieve). The extractor uses the answers of the oracle machine and extracts in polynomial-time the values that had to be proven knowledge of by the prover. That shows that the prover had to know the values beforehand in order to convince the verifier with non-negligible probability.

To prove a protocol to be zero-knowledge, one has to prove that the verifier has a *simulator* that, given the fact that the statement is proven and given the values received by the verifier, can simulate a transcript of the protocol that is indistinguishable of a transcript from a real interaction between the prover and the verifier.

Instead of being perfectly indistinguishable, the transcripts can be statistically or computationally indistinguishable. Then we call these proofs *statistical* cq. *computational zero-knowledge*.

Definition 4.1.3. *If knowledge of certain values is proven by a zero-knowledge proof, we call that proof a* zero-knowledge proof of knowledge.

Zero-knowledge proofs require robustness against arbitrary behavior by adversaries which can deviate from the protocol. However, a simpler type is a *honest verifier* adversary. The verifier then follows the protocol, but at the end he (or an eavesdropper) may have a transcript.

There are techniques to transform an honest verifier zero-knowledge protocol into a general zero-knowledge protocol (for example, see [Dam00]).

For precise, but nevertheless very readable introductions on zero-knowledge proofs of knowledge, we refer to [Gol04] and [DN07].

4.2 Ali Baba's Cave

An example of a zero-knowledge proof of knowledge

A clarifying example of a zero-knowledge proof of knowledge is given by Quisquater and others in an article called "How to Explain Zero-Knowledge Protocols to Your Children" [QGB90].

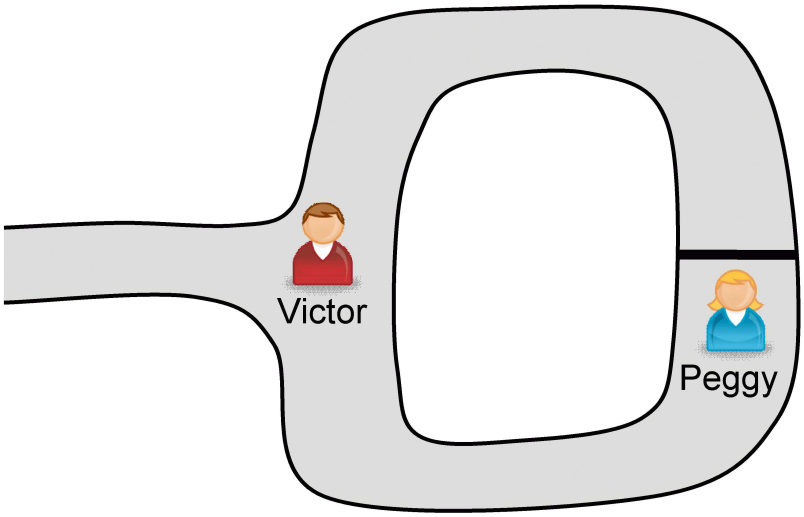

Figure 4.1: *Ali Baba's Cave.* Peggy has chosen a side and Victor has walked to the fork. There he shouts from which direction Peggy has to come.

We consider Ali Baba's circular cave of which Peggy (the prover) knows the magic word to open the door in the middle, as shown in Figure 4.1. Assume that Peggy wants to prove her knowledge of the magic word to Victor (the verifier), but she does not want to give it away.

A solution to this problem is as follows: Victor waits outside the cave, so he cannot see in which direction Peggy goes. After a few seconds Victor also goes into the cave and walks to the fork. There he shouts to Peggy from which side she has to come. If she really knows the magic word, she always succeeds, otherwise she succeeds only with probability 1/2. Repeating this several times reduces the probability of success when she does not know the word.

4.3 Zero-knowledge proofs in groups with hidden order

In 1989 an identification protocol based on the discrete logarithm was given by Schnorr [Sch89]. This protocol can be used as a zero-knowledge proof of knowledge of a discrete logarithm. The protocol is given in figure 4.2.

Common inputs: a large prime p;
a generator g of the cyclic group $\mathbb{Z}_p^*$
of order $q = p - 1$;
a public key X of the prover;

Prover's inputs: x such that $g^x = X \bmod p$.

$P \rightarrow V$ Choose $r \in [0, q)$ at random.
Send $R = g^r \bmod p$ to the verifier.
$P \leftarrow V$ Choose an integer challenge c from $[0, q)$ at random.
Send c to the prover.
$P \rightarrow V$ Compute the response $s = r + cx \bmod q$.
Send s to the verifier.

Acceptance: The verifier accepts if $RX^c = g^s \bmod p$.

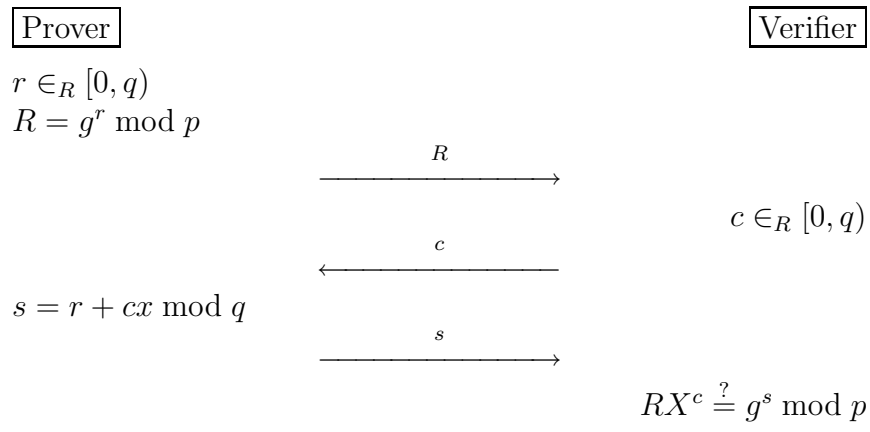

Figure 4.2: Schnorr Protocol to proof knowledge of a discrete logarithm

The Schnorr protocol is a simple and elegant method to prove knowledge of a discrete log, but unfortunately cannot be used in *idemix*: the order of the group Z_n^* (with n a special RSA-modulus) is unknown to the prover.

In 2002, Damgård and Fujisaki [DF02] presented a commitment scheme and a protocol to prove how to open a commitment based on the discrete logarithm on groups with hidden order. This is an improved version of a scheme suggested by Fujisaki and Okamoto [FO97]. The commitment scheme is statistically-hiding and the protocol is honest verifier statistical zero-knowledge. The set-up phase, commitment scheme and the proof of knowledge of commitment opening are slightly adapted to the situation in *idemix*, and are shown in figures 4.3, 4.4 and 4.5.

Set-Up
The verifier sets up the public key as follows:
he selects a group $\mathbb{Z}_n^*$ where $n = (2p' + 1)(2q' + 1)$ is a special RSA modulus and chooses a random elements $h \in QR_n$.
He checks if $\text{ord}(h) = p'q'$.
He then chooses $\alpha \in_R [0, p'q')$ and calculates $g = h^\alpha \in \langle h \rangle$.
The public key of the verifier is (n, g, h).

To prevent adversarial behaviour, the verifier must prove existence of the discrete logarithm $\log_h g \bmod n$. This can be done by a proof of knowledge with binary challenges, as described in [DF02].

Figure 4.3: Set-Up of the public key for the (adapted) Damgård-Fujisaki commitment scheme

Committing to a value
T is a system parameter that gives the interval for the choice of commitments, B is an estimation of the upper bound of the order of $\mathbb{Z}_n^*$, thus $\text{ord}(\mathbb{Z}_n^*) \leq 2^B$.
To make a commitment to a value $x \in [-T, T]$, the prover chooses $y \in_R [0, 2^{B+\ell_n}]$ and sets the commitment

$$X = g^x h^y \bmod n.$$

Because y is chosen with bit length at least $\ell_n + \log(\text{ord}(h))$, X is statistically close to uniform in $\langle h \rangle$ for any value of x.

Figure 4.4: Making a commitment with the (adapted) Damgård-Fujisaki commitment scheme

Common inputs: security parameter ℓ_c defining the challenge size;
a public key (n, g, h) of the verifier;
a commitment $X = g^x h^y \bmod n$;
parameters B and T as given in Figure 4.4.
Prover's inputs: x and y.

$P \to V$ Choose integers $r_1 \in [0, 2^{\ell_n+\ell_c}T)$
and $r_2 \in [0, 2^{B+2\ell_n+\ell_c})$ at random.
Send $R = g^{r_1}h^{r_2} \bmod n$ to the verifier.
$P \leftarrow V$ Choose an integer challenge c from $[0, 2^{\ell_c})$ at random.
Send c to the prover.
$P \to V$ Compute the response $s_1 = r_1 + cx$ and $s_2 = r_2 + cy \in \mathbb{Z}$.
Send s_1 and s_2 to the verifier.

Acceptance: The verifier accepts if $RX^c = g^{s_1}h^{s_2} \bmod n$.

Prover		Verifier
$r_1 \in_R [0, 2^{\ell_n+\ell_c}T)$		
$r_2 \in_R [0, 2^{B+2\ell_n+\ell_c})$		
$R = g^{r_1}h^{r_2} \bmod n$		
	$\xrightarrow{R}$	
		$c \in_R [0, 2^{\ell_c})$
	$\xleftarrow{c}$	
$s_1 = r_1 + cx \in \mathbb{Z}$		
$s_2 = r_2 + cy \in \mathbb{Z}$		
	$\xrightarrow{s_1, s_2}$	
		$RX^c \stackrel{?}{=} g^{s_1}h^{s_2} \bmod n$

Figure 4.5: (Adapted) Damgård-Fujisaki proof of knowledge of commitment opening

4.4 Zero-knowledge under concurrent composition

To use zero-knowledge proofs of knowledge in practical situations, it is necessary that they preserve security even under concurrent composition. Here (polynomially) many instances of the proofs are invoked at arbitrary times and proceed at arbitrary speed.

However, zero-knowledge of proofs of knowledge is proven by showing that there exists a simulator for the verifier that can simulate a transcript of the protocol. Even if a simulation can be done for a single execution of the protocol, it is not necessarily possible for the concurrent version. In fact, there are examples of proofs that lose their zero-knowledge property under concurrent composition.

Damgård has shown [Dam00] that some honest-verifier zero-knowledge protocols, namely the Σ-protocols, can be zero-knowledge even under concurrent composition. Such a Σ-protocol is characterised by the three steps *commitment*, *challenge* and *response*, like the previous protocols we have shown.

To make a protocol of that form secure under concurrent composition, we need a *witness hiding* commitment scheme. In that case the commitment is binding to a value for the prover, but the verifier possesses a trapdoor which allows the verifier to open that commitment to an arbitrary value of its choosing (ie. cheating by pretending to have committed to this value), which is called *witness indistinguishability*.

All protocols presented below are Σ-protocols using a witness hiding commitment scheme, and thus are concurrent composable.

4.5 Zero-knowledge building blocks in *idemix*

There are three basic building blocks to construct all proofs of knowledge needed in *idemix*:

1. Proof of knowledge of a discrete logarithm representation modulo a composite,
   denoted by
   $$\mathrm{PK}\left\{ (\alpha_1, \ldots, \alpha_u) : y = g_1^{\alpha_1} \cdots g_u^{\alpha_u} \right\}.$$

2. Proof of knowledge of equality of discrete logarithms on different bases or in different groups,
   denoted by
$$\mathrm{PK}\left\{ (\alpha, \beta, \gamma) : y_1 = g_1^\alpha h_1^\beta \wedge y_2 = g_2^\alpha h_2^\gamma \right\}.$$
3. Proof that a discrete logarithm lies in an interval,
   denoted by
$$\mathrm{PK}\left\{ (\alpha) : y = g^\alpha \wedge \alpha \in [a, b] \right\}.$$

Details about the notation can be found in section 3.3.1.

4.5.1 Proof of knowledge of commitment opening

In her PhD thesis [Lys02] Anna Lysyanskaya presents a Σ-Protocol Zero-knowledge Proof of Representation Modulo a Composite. She proves that it is really a Σ zero-knowledge protocol. Unfortunately she shows only the protocol and its proof for the basis g and remarks: It is easy to see how to generalise this protocol and the corresponding extractor to the case where instead of g, there are several bases $(g_1, \ldots, g_u)$.

In this section we see how to generalise her protocol and its proof for discrete logarithm representations with several bases.

We now show a proof of knowledge protocol for proving knowledge of a committed value. We give a Σ-protocol proof of knowledge of representation: it can be converted into a general zero-knowlegde proof using standard techniques based on trapdoor commitment schemes, described in Section 2.6.1 of Lysyanskaya's PhD thesis. All protocols are in the public-parameter model.

Lemma 4.5.1. *Under the Strong RSA assumption, the protocol described in Figure 4.6 is a Σ-protocol zero-knowledge proof of knowledge of witness* $(\vec{x}, y) := (x_1, \ldots, x_u, y)$ *for the relation*
$$R_{n,g_1,\ldots,g_u,h} = \{X, (\vec{x}, y) \ : \ X = g_1^{x_1} \cdots g_u^{x_u} h^y \bmod n\}.$$

Proof. Let $X = g_1^{x_1} \cdots g_u^{x_u} h^y \bmod n$ as in the protocol described in Figure 4.6. We have to prove completeness, soundness en zero-knowledge of the protocol.

Completeness. The completeness is obvious: if the prover follows the protocol, then
$$\begin{aligned} RX^c &= g_1^{r_1} \cdots g_u^{r_u} h^{r_{u+1}} (g_1^{x_1} \cdots g_u^{x_u} h^y)^c \\ &= g_1^{r_1 + cx_1} \cdots g_u^{r_u + cx_u} h^{r_{u+1} + cy} \\ &= g_1^{s_1} \cdots g_u^{s_u} h^{s_{u+1}}. \end{aligned}$$

Common inputs: security parameters ℓ_r and ℓ_c;
$(n, g_1, \ldots, g_u, h)$: a public key of the commitment,
where n is a special RSA modulus;
a commitment $X = g_1^{x_1} \cdots g_u^{x_u} h^y \bmod n$.
Prover's inputs: $x_1, \ldots, x_u$ and y.

$P \to V$ Choose ℓ_r-bit $r_1, \ldots, r_{u+1}$ at random.
Send $R = g_1^{r_1} \cdots g_u^{r_u} h^{r_{u+1}} \bmod n$ to the verifier.
$P \leftarrow V$ Generate an ℓ_c-bit challenge c.
$P \to V$ Take $x_{u+1} = y$.
Compute the response $s_1 = r_1 + cx_1, \ldots, s_{u+1} = r_{u+1} + cx_{u+1}$.

Acceptance: The verifier accepts if $RX^c = g_1^{s_1} \cdots g_u^{s_u} h^{s_{u+1}} \bmod n$.

Prover		Verifier
$r_1, \ldots, r_{u+1} \in_R [0, 2^{\ell_r})$ $R = g_1^{r_1} \cdots g_u^{r_u} h^{r_{u+1}} \bmod n$		
	$\xrightarrow{R}$	
		$c \in_R [0, 2^{\ell_c})$
	$\xleftarrow{c}$	
$s_1 = r_1 + cx_1 \in \mathbb{Z}$		
$\vdots$		
$s_{u+1} = r_{u+1} + cx_{u+1}$ $= r_{u+1} + cy \in \mathbb{Z}$		
	$\xrightarrow{s_1, \ldots, s_{u+1}}$	
		$RX^c \stackrel{?}{=} g_1^{s_1} \cdots g_u^{s_u} h^{s_{u+1}} \bmod n$

Figure 4.6: Σ-Protocol Proof of Representation Modulo a Composite

Zero-knowledge. The Σ zero-knowledge property follows because suppose the challenge c is fixed in advance. Then the simulator chooses $s_1, \ldots, s_{u+1}$ at random, and sets $R = g_1^{s_1} \cdots g_u^{s_u} h^{s_{u+1}} / X^c$. If $\ell_r = \ell_c + \max(\ell_x, \ell_y, \ell_n) + \ell$ where ℓ is a security parameter, then choosing $s_1, \ldots, s_{u+1}$ at random is statistically indistinguishable from the responses he recieves from the real prover. Therefore, the verifier cannot distinguish the responses of the prover from just randomly picked values and thus the verifier gets no information about the values which are hidden in X.

Soundness. The knowledge extraction follows because suppose that with non-negligible probability over the choice of the challenge c, the prover causes the verifier to accept. Then the verifier can get two accepting transcripts based on the same R, with challenges c and c'. This results in the equations $RX^c = g_1^{s_1} \cdots g_u^{s_u} h^{s_{u+1}}$ and $RX^{c'} = g_1^{s'_1} \cdots g_u^{s'_u} h^{s'_{u+1}}$.

Claim 4.5.2. *Let $\delta = c - c'$. Then $\delta \mid (s_i - s'_i)$ for all $1 \leq i \leq u+1$ under the Strong RSA assumption.*

Proof of claim. Suppose that (n, g) is an instance of the Flexible RSA problem, with n a special RSA modulus and g a generator of QR_n. The extractor sets up the public key of the commitment by letting $\alpha_1 = 1$ and choosing random α_i's for $2 \leq i \leq u+1$ of length $\ell_s + 3$ and letting the public key $PK = (n, g_1, \ldots, g_u, h)$, where $g_i = g^{\alpha_i} \bmod n$ for $1 \leq i \leq u$ and $h = g^{\alpha_{u+1}} \bmod n$. This PK is indistinguishable from a random PK since the α_i's are random and g is a generator.

Now the extractor obtains equations $RX^c = g_1^{s_1} \cdots g_u^{s_u} h^{s_{u+1}}$ and $RX^{c'} = g_1^{s'_1} \cdots g_u^{s'_u} h^{s'_{u+1}}$. Let $S = \sum_{i=1}^{u+1} \alpha_i (s_i - s'_i)$, $\delta = c - c'$, and suppose for contradiction that $\delta \nmid (s_k - s'_k)$ for some k or that $\delta \nmid S$.

The equations give us that $X^\delta \equiv g^S \bmod n$. Therefore, if $\delta \nmid S$, we break the Strong RSA assumption by Lemma 2.4.3 and we are done.

So we assume that $\delta \mid S$ and $\delta \nmid (s_k - s'_k)$ for some k. Our goal is now to show that this case happens with probability at most $1/2$.
Note that if $\delta \mid (s_i - s'_i)$ for $2 \leq i \leq u+1$, it follows that $\delta \mid (s_1 - s'_1)$. Thus we assume that there exists some $k \geq 2$ such that $\delta \nmid (s_k - s'_k)$. Now select such a k.

For $2 \leq i \leq u+1$ we write

$$\alpha_i = \beta_i + \gamma_i \cdot p'q' \quad \text{with } 0 \leq \beta_i < p'q'.$$

Note that β_i is uniquely determined as the smallest exponent such that $g^{\alpha_i} \equiv g^{\beta_i} \bmod n$. Our assumption that $\delta \mid S$ gives us

$$S = (s_1 - s_1') + \sum_{i=2}^{u+1} \beta_i \ (s_i - s_i') + p'q' \sum_{i=2}^{u+1} \gamma_i \ (s_i - s_i') \equiv 0 \bmod \delta.$$

Note that from the extractor's point of view, the γ_i's are chosen uniformly at random from at least $2^{\ell_s+3}/p'q'$ values, and must satisfy the above equation. By our assumption, this equation has at least one solution, and now we count the number of solutions.

First, take $(x_2, \ldots, x_{u+1}) := (\gamma_2, \ldots, \gamma_{u+1}) \bmod \delta$ as a particular solution of

$$(s_1 - s_1') + \sum_{i=2}^{u+1} \beta_i \ (s_i - s_i') + p'q' \sum_{i=2}^{u+1} x_i \ (s_i - s_i') \equiv 0 \bmod \delta.$$

Then the general solution is the set of tuples $(\gamma_2 + \epsilon_2, \ldots, \gamma_{u+1} + \epsilon_{u+1})$ with $(\epsilon_2, \ldots, \epsilon_{u+1}) \in (\mathbb{Z}/\delta\mathbb{Z})^u$ in the subgroup H given by

$$\sum_{i=2}^{u+1} \epsilon_i \ (s_i - s_i') \equiv 0 \bmod \delta.$$

Because $(s_k - s_k') \not\equiv 0 \bmod \delta$, we know that $(\epsilon_2 = 0, \ldots, 0, \epsilon_k = 1, 0, \ldots, \epsilon_{u+1} = 0) \notin H$ and therefore H does not contain all elements of $(\mathbb{Z}/\delta\mathbb{Z})^u$. So H is a strict subgroup of $(\mathbb{Z}/\delta\mathbb{Z})^u$ and has index at least 2. Thus the γ_i's satisfy the equation with probability at most $1/2$.

So, if $\delta \nmid (s_k - s_k')$ for some $1 < k < u + 1$, we find with non-neglible probability a solution to our instance of the Flexible RSA problem, which contradicts the Strong RSA assumption. □

From the claim, it follows that, if we let $x_i = (s_i - s_i')/\delta$ for $1 \leq i \leq u$ and $y = (s_{u+1} - s_{u+1}')/\delta$, then $(x_1, \ldots, x_u, y)$ is a representation of X in bases $g_1, \ldots, g_u$ and h modulo n. Note that the extracted values are not necessarily positive. □

4.5.2 Proof of equality

We show a proof of knowledge protocol for proving knowledge and equality of two or more committed values. We give a Σ-protocol proof of knowledge of equality.

Lemma 4.5.3. *Under the Strong RSA assumption, the protocol described in Figure 4.7 is a Σ-protocol zero-knowledge proof of knowledge of witness* (x, y_1, y_2) *for the relation*

$$R_{n_1,g_1,h_1,n_2,g_2,h_2} = \{X_1, X_2, (x, y_1, y_2) \ : \ X_1 = g_1^x h_1^{y_1} \bmod n_1 \ \wedge \ X_2 = g_2^x h_2^{y_2} \bmod n_2 \ \}.$$

Proof.

Completeness and **Zero-knowledge** are proven in the same way as for Lemma 4.5.1.

Soundness. Let the verifier obtain two accepting transcripts based on the same R, with challenges c and c'. If we use Claim 4.5.2 with $\delta = c - c'$ it follows that $x = (s_1 - s_1')/\delta$ and $y_1 = (s_{21} - s_{21}')/\delta$ give a representation of X_1 and x and $y_2 = (s_{22} - s_{22}')/\delta$ give a representation of X_2. □

4.6 Zero-knowledge interval proofs

This section describes various ways of checking that a committed number lies in a certain interval. The first observation we will make is that executing the Damgård-Fujisaki protocol, or in our situation the Σ-protocol Proof of Representation, already proves that the committed number x_1 lies in a larger, but known interval (as mentioned in [DF02]). After that we look at the exact interval proof of Boudot. And finally we will show that the less precise results that directly follow from executing the Σ-protocol Proof of Representation are sufficient to guarantee security in *idemix*.

4.6.1 Proof that a committed number lies in an expanded interval

Assume that the prover chooses the hidden number x_1 from the interval $I = [-T, T] = [-2^t, 2^t]$. Also modify the proof of representation (as described in Figure 4.6) so that $r_1 \in [0, 2^{\ell_c+\ell}T)$, with ℓ a security parameter as decribed in section 3.5. Then after executing the proof of knowledge of the representation, the verifier will be convinced that x_1 lies in an interval J, where the expansion rate $|J|/|I|$ is equal to $2^{\ell_c}(2^\ell + 2)$, i.e., the interval J equals

$$\left(-2^{\ell_c}T(2^\ell + 2),\ 2^{\ell_c}T(2^\ell + 2)\right).$$

Common inputs: security parameters ℓ_r and ℓ_c;
(n_1, g_1, h_1) and (n_2, g_2, h_2) public keys
of the verifier for the commitments,
where n_1 and n_2 are special RSA moduli;
commitments $X_1 = g_1^x h_1^{y_1} \bmod n_1$ and $X_2 = g_2^x h_2^{y_2} \bmod n_2$.
Prover's inputs: x, y_1 and y_2.

$P \to V$ Choose ℓ_r-bit r_1, r_{21}, r_{22} at random.
Send $R_1 = g_1^{r_1} h_1^{r_{12}} \bmod n$ and $R_2 = g_2^{r_1} h_2^{r_{22}} \bmod n$ to the verifier.
$P \leftarrow V$ Generate an ℓ_c-bit challenge c.
$P \to V$ Compute the response $s_1 = r_1 + cx, s_{21} = r_{21} + cy_1, s_{22} = r_{22} + cy_2$.

Acceptance: The verifier accepts if $R_1 X_1^c = g_1^{s_1} h_1^{s_{21}} \bmod n_1$ and $R_2 X_2^c = g_2^{s_1} h_2^{s_{22}} \bmod n_2$.

Prover		Verifier
$r_1, r_{21}, r_{22} \in_R [0, 2^{\ell_r})$ $R_1 = g_1^{r_1} h_1^{r_{12}} \bmod n_1$ $R_2 = g_2^{r_1} h_2^{r_{22}} \bmod n_2$		
	$\xrightarrow{R_1, R_2}$	
		$c \in_R [0, 2^{\ell_c})$
	$\xleftarrow{c}$	
$s_1 = r_1 + cx \in \mathbb{Z}$ $s_{21} = r_{21} + cy_1 \in \mathbb{Z}$ $s_{22} = r_{22} + cy_2 \in \mathbb{Z}$		
	$\xrightarrow{s_1, s_{21}, s_{22}}$	
		$R_1 X_1^c \stackrel{?}{=} g_1^{s_1} h_1^{s_{21}} \bmod n_1$ $R_2 X_2^c \stackrel{?}{=} g_2^{s_1} h_2^{s_{22}} \bmod n_2$

Figure 4.7: Σ-Protocol Proof of Equality

This can be seen as follows:
After executing the proof of representation with x_1 of the commitment and r_1 of the first step of the protocol from the correct intervals, the value of $s_1 = r_1 + cx_1$ lies in $\left(-2^{\ell_c}T, 2^{\ell_c}T(2^\ell + 1)\right)$. We can check this in the last step of the protocol. That does not tell us that x_1 is in the right interval, but we will explain that it tells us something about a larger interval x_1 is in.

From the soundness of the proof of representation it follows that s_1 has to be of the form $s_1 = r_1 + cx_1$, so we can use that here. If the verifier uses 0 as a challenge, than he gets no information about x_1, so we assume that $c \geq 1$.
Using the information that $s_1 > -2^{\ell_c}T$ gives us

$$\min x_1 = \min \frac{s_1 - r_1}{c} \geq \min(s_1 - r_1) > -2^{\ell_c}T - 2^{\ell_c+\ell}T = -2^{\ell_c}T(2^\ell + 1) \ ,$$

while using that $s_1 < 2^{\ell_c}T(2^\ell + 1)$ gives us

$$\max x_1 = \max \frac{s_1 - r_1}{c} \leq \max(s_1 - r_1) < 2^{\ell_c}T(2^\ell + 1) \ .$$

In the mentioned equations we used that r_1 is chosen from the correct interval, but at this point we are not sure about that. We investigate what happens if the prover succeeds the proof of representation with values r_1 or x_1 outside the correct intervals. Assume that the prover dishonestly proves that x_1 is in the interval $\left(-2^{\ell_c}T(2^\ell + 1), 2^{\ell_c}T(2^\ell + 1)\right)$ while actually $x_1 > 2^{\ell_c}T(2^\ell + 1)$. Then the prover has a triple (r_1, c, s_1) satisfying the equation $s_1 = r_1 + cx_1$, lying in the interval $\left(-2^{\ell_c}T, 2^{\ell_c}T(2^\ell + 1)\right)$. Now assume that, instead of c, the challenge was chosen $\tilde{c} = c + a$, $a > 0$. Then $\tilde{s_1}$ has to be

$$\tilde{s_1} = r_1 + \tilde{c}x_1 = s_1 + ax_1 > -2^{\ell_c}T(2^\ell + 1) + a \cdot 2^{\ell_c}T(2^\ell + 1) = (a-1)2^{\ell_c}T(2^\ell + 1).$$

Or assume that, instead of c, the challenge was chosen $\bar{c} = c - a$, $a > 0$. Then $\bar{s_1}$ has to be

$$\bar{s_1} = r_1 + \bar{c}x_1 = s_1 - ax_1 < 2^{\ell_c}T(2^\ell + 1) - a \cdot 2^{\ell_c}T(2^\ell + 1) = (1-a)2^{\ell_c}T(2^\ell + 1).$$

For both challenges it holds that if $a \geq 2$, the resulting $\tilde{s_1}$ or $\bar{s_1}$ will lie outside the interval. The same argument can be used if $x_1 < -2^{\ell_c}T(2^\ell + 1)$. If $|x_1| > 2^{\ell_c}T(2^\ell + 2)$, then the given r_1 only satisfies $s_1 = r_1 + cx_1$ for a unique choice of c. Because the probability of guessing c is assumed to be negligible, the prover cannot prove that x lies in the correct interval if x_1 actually is greater than $2^{\ell_c}T(2^\ell + 2)$.

Thus the verifier is convinced that x_1 lies in the interval J.

4.6.2 The interval proof of Boudot

The following proof is due to Boudot [Bou00] and simplified to the case of a symmetric interval.

We have as input a commitment $C = g^x h^r \bmod n$ to an integer x and now we want to prove the knowledge of x and that x lies in the interval $[-W, W] = [-2^w, 2^w]$.

To show that $x \in [-W, W]$ is the same as to prove that $W^2 - x^2 \geq 0$, so $W^2 - x^2 > -1$.

We begin by briefly sketching how this is achieved.

First we commit to x^2 with commitment D and prove that it really is a commitment to the square of x. We calculate $D_W = g^{W^2}/D$ and have to prove that D_W hides a value $\tilde{x} = W^2 - x^2$ equal to or greater than zero. Now Boudot suggests to scale this up, so we set $\bar{D} = D_W^{2^S}$ for a certain value S. Also multiply $\tilde{x}$ and $\tilde{r}$ with 2^S to obtain $\bar{x}$ and $\bar{r}$. Now we have to prove that $\bar{D}$ hides a secret $\bar{x}$ greater than -2^S. First we write $\bar{x}$ as the sum of the biggest possible square plus a positive small number, ie. $\bar{x} = (\bar{x}_1)^2 + \bar{x}_2$. Then we hide the values $(\bar{x}_1)^2, \bar{x}_2$ in the commitments $\bar{E}_1, \bar{E}_2$ and show the following three facts (indirectly or directly) to Bob:

1. The sum of what $\bar{E}_1$ and $\bar{E}_2$ hide is what $\bar{D}$ hides.
2. $\bar{E}_1$ hides a square, thus a positive value.
3. $\bar{E}_2$ hides a value with absolute value smaller than 2^S.

In the previous paragraph we provided an explanation in words of the Boudot proof. In this section this will be worked out in detail, so that we can see what calculation power is needed.

1. *Commitment to x^2 and proving it*
   The prover chooses $r_1 \in_R [0, 2^{\ell_r})$ and computes $r_2 = r_1 - rx \in \mathbb{Z}$ and $D = C^x h^{r_2} \bmod n$. Note that $D = g^{x^2} h^{rx+r_2} = g^{x^2} h^{r_1} \bmod n$. To prove that D hides the square of what C hides, we execute:
   $$\mathrm{PK}\left\{ (\alpha, \beta, \gamma) : C = g^\alpha h^\beta \wedge D = C^\alpha h^\gamma \right\}.$$
2. *Transposing*
   We have to prove $\tilde{x} = W^2 - x^2 \geq 0$, so both parties calculate $D_W = g^{W^2}/D (= g^{W^2 - x^2} h^{-r_1})$. We define $\tilde{r} = -r_1 \in (-2^{\ell_r}, 0]$. The prover has to convince the verifier that D_W hides a secret equal to or greater than zero.

3. *Scaling up*
   The basic Boudot proof only gives certainty for $\tilde{x}$ lying in an interval $(-\theta, \infty)$. If we scale it up so that $-\theta > -1$, we prove that $\tilde{x} \geq 0$, because $\tilde{x}$ is an integer. We scale D_W and $\tilde{x}$ by a factor 2^S, where $S = 2(\ell_c + \ell + 2)$, thus both calculate $\bar{D} = D_W^{2^S}$ and the prover also calculates $\bar{x} = 2^S\tilde{x}$ and $\bar{r} = 2^S\tilde{r}$. Now we have to prove that $\bar{D}$ hides a secret $\bar{x} > -2^S$.

4. *Decomposition of $\bar{x}$*
   The prover computes

   $$\bar{x}_1 = \lfloor\sqrt{\bar{x}}\rfloor, \quad \bar{x}_2 = \bar{x} - \bar{x}_1^2.$$

   Computing the floor of a square root can be done efficiently by using Newton's Method, see Appendix A. Note that $\bar{x}$ is decomposed into a square and a small value ($\bar{x} = \bar{x}_1^2 + \bar{x}_2$) where $\bar{x}_2$ satisfies

   $$0 \leq \bar{x}_2 \leq \bar{x} - (\sqrt{\bar{x}} - 1)^2 < 2\sqrt{\bar{x}} = 2\sqrt{2^S\tilde{x}} \leq 2\sqrt{2^S W^2} = 2^{(S/2)+1}W.$$

5. *Computation of new commitments*
   The prover selects random values $\bar{r}_1, \bar{r}_2 \in [0, 2^{\ell_r})$ such that $\bar{r}_1 + \bar{r}_2 = \bar{r}$. He computes

   $$E_1 = g^{\bar{x}_1^2} h^{\bar{r}_1} \bmod n, \quad E_2 = \bar{D}/E_1 \bmod n$$

   He sends E_1 to the verifier who also calculates $E_2 = \bar{D}/E_1 \bmod n$. Note that $E_2 = g^{\bar{x}} h^{\bar{r}} / g^{\bar{x}_1^2} h^{\bar{r}_1} = g^{\bar{x}_2} h^{\bar{r}_2} \bmod n$.

6. *Validating the commitment to a square*
   The prover and the verifier execute

   $$\mathrm{PK}\left\{ (\alpha, \beta) : E_1 = g^{\alpha^2} h^{\beta} \right\}.$$

   This can be done as follows:
   The prover chooses $r' \in_R [0, 2^{\ell_r})$, computes $F = g^{\bar{x}_1} h^{r'}$ and sends F to the verifier. After that they execute

   $$\mathrm{PK}\left\{ (\alpha, \beta, \gamma) : F = g^{\alpha} h^{\beta} \wedge E_1 = F^{\alpha} h^{\gamma} \right\}.$$

7. *Validating the commitment to a small value*
   The prover and the verifier execute the Σ-protocol proof of representation:

   $$\mathrm{PK}\left\{ (\alpha, \beta) : E_2 = g^{\alpha} h^{\beta} \right\}.$$

Because $\bar{x}_2 \in [0, 2^{S/2+1}W)$, after executing the proof of knowledge the verifier can check that

$$\begin{aligned} \bar{x}_2 &\in \left(-2^{\ell_c}(2^{\ell}+2)2^{(S/2)+1},\ 2^{\ell_c}(2^{\ell}+2)2^{(S/2)+1}\right) \\ &= \left(-2^{\ell_c+\ell+1+(S/2)+1},\ 2^{\ell_c+\ell+1+(S/2)+1}\right) \\ &= \left(-2^{(\ell_c+\ell+2)+(S/2)},\ 2^{(\ell_c+\ell+2)+(S/2)}\right) = \left(-2^{S},\ 2^{S}\right), \end{aligned}$$

what had to be proven.

Note: If the interval is not symmetric, it can be made symmetric by translation. The original version of the Boudot proof uses another method of proving that $x \in [a, b]$: it proves that values hidden by $C_a = g^a/C = g^{a-x}h^r$ and $C_b = C/g^b = g^{x-b}h^{-r}$ are positive numbers.

Analysis

To analyse the computing load of this interval proof, we again look at the load at the prover's side.
All exponentations are done modulo n of bitlength ℓ_n. We also mention the size of the exponent for each exponentiation.

1. Calculation of $D = C^x h^{r_2} = g^{x^2} h^{r_1} \bmod n$, *2 exponentiations* with exponents $x^2 \in [0, W^2]$ and $r_1 \in [0, 2^{\ell_r})$.
   The proof of knowledge: *4 exponentiations* with exponents in $[0, 2^{\ell_r})$.

2. Calculation of D_W is not necessary for the prover, knowledge of the exponents $\tilde{x}$ and $\tilde{r}$ is sufficient.

3. Calculation of $\bar{D}$ is not necessary for the prover, knowledge of the exponents $\bar{x}$ and $\bar{r}$ is sufficient.

5. Calculation of $E_1 = g^{\bar{x}_1^2} h^{\bar{r}_1} \bmod n$, *2 exponentiations* with exponents $\bar{x}_1^2 \in [0, 2^S W^2]$ and $\bar{r}_1 \in [0, 2^{\ell_r})$. Calculation of E_2 is not necessary for the prover, knowledge of the exponents $\bar{x}_2$ and $\bar{r}_2$ is sufficient.

6. Calculation of $F = g^{\bar{x}_1} h^{r'} \bmod n$, *2 exponentiations* with exponents $\bar{x}_1 \in [0, 2^{(S/2)}W)$ and $r' \in [0, 2^{\ell_r})$.
   The proof of knowledge: *4 exponentiations* with exponents in $[0, 2^{\ell_r})$.

7. The proof of knowledge: *2 exponentiations* with exponents in $[0, 2^{\ell_r})$.

So we see that we have to do *16 exponentiations*. Of that exponentiations, 13 have a ℓ_r-bit exponent, 1 has a $2w$-bit exponent, 1 has a $2(w+\ell_c+\ell+2)$-bit exponent, 1 exponentiation has an approximately $(w+\ell_c+\ell+2)$-bit exponent. If we take $w = \ell_x - 1 = 159$ and $\ell_c = 80$, we obtain:

$$\begin{aligned} \ell_r &= \ell_c + \ell_s + \ell = 80 + 2386 + 160 = 2626 \\ 2w &= 318 \\ w + \ell_c + \ell + 2 &= 159 + 80 + 160 + 2 = 402 \\ 2(w + \ell_c + \ell + 2) &= 804 \end{aligned}$$

As described in appendix B, a 2048-bit exponentiation takes approximately 174 ms. And because the time a exponentiation takes depends quadratically on the bitlenght, we see that a Boudot interval proof on the user's master key takes about $(13 \cdot (2626/2048)^2 + (318/2048)^2 + (804/2048)^2 + (402/2048)^2) \cdot 174 \text{ ms} \approx 4$ seconds.

4.6.3 A smart way of checking intervals in *idemix*

The analysis in the previous section shows us that a Boudot interval proof needs a huge amount of calculation power. It is however possible to garantuee security in *idemix* without doing explicit interval proofs.

The expanded interval proof from Section 4.6.1 can be used to give a sufficient proof of security in *idemix*. For security of the signing algorithm, we require that the value of x_U lies in $[a, b]$. So we ask the user to choose an x_U in a smaller interval such that after executing the zero-knowledge proof it certainly is showed that $x \in [a, b]$.

Assume we execute the proof of representation with $x_U \in [0, 2^{\ell_x})$ from the pseudonym and a random value $r \in [0, 2^{\ell_c+\ell+\ell_x})$ in the first step of the protocol. Then the value of $s = r + cx_U$ lies in the interval $[0, 2^{\ell_c+\ell_x}(2^\ell + 1))$.

We can easily see that after executing the proof representation, the verifier is convinced of the fact that

$$x_U \in \left(-2^{\ell_c+\ell_x+\ell},\ 2^{\ell_c+\ell_x}(2^\ell + 1)\right).$$

This is the same as saying that the verifier is convinced that

$$x_U + 2^{\ell_c+\ell_x+\ell} \in \left(0,\ 2^{\ell_c+\ell_x}(2^{\ell+1} + 1)\right).$$

A translation of x_U does not affect the security of the signing algorithm. So if we prove security of the signature for the message space

$$\left(0,\ 2^{\ell_c+\ell_x}(2^{\ell+1}+1)\right) \subset \left(0,\ 2^{\ell_c+\ell_x+\ell+2}\right) = \left(0,\ 2^{\ell_m}\right) \subset \left[0,\ 2^{\ell_m}\right),$$

and if the prover has $x_U \in [0, 2^{\ell_x})$, the verifier is always convinced that the prover owns a secure signature.

To have an effective message space of 160 bits, we thus have to choose $\ell_m = \ell_c + \ell_x + \ell + 2 = 80 + 160 + 80 + 2 = 322$.

In the same way we can achieve that the exponents e_{UO} lie in the interval $[2^{\ell_e-1},\ 2^{\ell_e})$. Namely, if the issuing organisation always chooses the signature exponent $e_{UO} \in [2^{\ell_e-1} + 2^{\ell_e-2} - 2^{\ell_e-\ell_c-\ell-3},\ 2^{\ell_e-1} + 2^{\ell_e-2} + 2^{\ell_e-\ell_c-\ell-3}]$, after execution of the proof of representation, the verifier will be convinced that $e_{UO} \in (2^{\ell_e-1}, 2^{\ell_e})$.

This can be done as follows: for the random exponent r_e corresponding to the exponent e_{UO} in the proof of representation, choose $r_e \in_R [0, 2^{\ell_e-3})$. Now instead of proving knowledge of e_{UO}, prove knowledge of $\tilde{e} := e_{UO} - (2^{\ell_e-1} + 2^{\ell_e-2})$. Note that $\tilde{e} \in [-2^{\ell_e-\ell_c-\ell-3},\ 2^{\ell_e-\ell_c-\ell-3}]$.

In the verification step of the protocol, the verifier checks that

$$\tilde{s} = r_e + c \cdot \tilde{e} \in \left(-2^{\ell_e-\ell-3}, 2^{\ell_e-3} + 2^{\ell_e-\ell-3}\right) = \left(-2^{\ell_e-\ell-3}, 2^{\ell_e-3}(2^{-\ell}+1)\right).$$

From this it follows that

$$\begin{aligned}
\tilde{e} &\in [\min(\tilde{s} - r_e),\ \max(\tilde{s} - r_e)] \\
&= \left(-2^{\ell_e-\ell-3} - 2^{\ell_e-3},\ 2^{\ell_e-3}(2^{-\ell}+1)\right) \\
&= \left(-2^{\ell_e-3}(2^{-\ell}+1),\ 2^{\ell_e-3}(2^{-\ell}+1)\right) \\
&\subset \left(-2^{\ell_e-2},\ 2^{\ell_e-2}\right).
\end{aligned}$$

So we obtain that

$$e_{UO} \in \left(2^{\ell_e-1},\ 2^{\ell_e-1} + 2^{\ell_e-2} + 2^{\ell_e-2}\right) = \left(2^{\ell_e-1},\ 2^{\ell_e}\right).$$

Chapter 5

Distribution of *idemix* on a smart card and terminal

As described in the first chapter, integrating a smart card in *idemix* has some advantages in security and portability. However, the limited computing power and storage capacities of the smart card put constraints on the feasibility of an implementation where all calculations are done on the smart card. A way to solve this is to use the resources on the terminal in which the smart card is inserted. In this chapter we will explore in which way the calculations and storage have to be distributed over the smart card and a terminal.

5.1 Distributing information and calculations

First we have to decide how to distribute the information and calculations over the smart card and the terminal. By analysing *idemix* we find that there are four blocks of information:

1. The user's master key x_U.
2. The pseudonym P_{UO} (or P_{UI}) of the user with an issuer.
3. The pseudonym P_{UV} of the user with a verifier.
4. The credential triple (c_{UO}, e_{UO}, r_{UO}).

The relations between these blocks restrict the possible choices for a distribution. For example, giving the master key x_U to the terminal makes it useless to hide pseudonyms or credentials which are bound to this key.

This leads to the following reasonable distributions:

1. **The smart card gives all information to the terminal.**
   In this case the smart card gives the user's master key, the needed pseudonyms and needed credentials to the terminal. Therefore, the terminal can do all calculations, the smart card is only a storage device.

2. **The smart card only keeps the master key secret.**
   Now the smart card gives the needed pseudonyms and credentials to the terminal, but keeps the user's master key secret. This implies that the terminal can do a lot of calculations with respect to the pseudonyms and credentials, but the calculations directly related to the master key have to be done on the smart card.

   The smart card only gives the needed credentials to the terminal.
   This leads to almost the same distribution as in the previous possibility, as the terminal can easily calculate the pseudonym from the credential (ie. $P_{UO} = c_{UO}^{e_{UO}} / d_O h_O^{r_{UO}}$).

3. **The smart card only gives the pseudonym with the verifier to the terminal.**
   This distribution is useful when the terminal is owned by the verifier, since the verifier obtains no linking information.

4. **The smart card keeps everything secret.**
   In this case all calculations have to be done by the smart card. The terminal is merely used as a communication device between the smart card and the organisation.

5.2 Analysis of distributions

In this section we will explore in detail the advantages and disadvantages of the mentioned distributions. We will also see how this effects the protocols from Chapter 3.

5.2.1 The smart card gives all information to the terminal

It is possible to give all information to the terminal. As the terminal has much more calculation power than the smart card, this will be the solution with fastest execution of the protocol. However, the smart card has no more use than being a portable storage device, because all 'secrets' are told to the terminal. Therefore, this is not what we are looking for.

5.2.2 The smart card only keeps the master key secret

This distribution is interesting: the most sensitive information, that is, the user's master key, remains secret. All other information is given to the terminal, so that all calculations which do not directly involve the master key can be done by the terminal.

Because the pseudonym of the user with a verifier as well as the pseudonym and credential with a certain issuer will be known by the terminal, this distribution leaks linking information to the terminal.

This distribution requires a modification of the functions VerifyCred and VerifyCredOnNym. We will treat this in Section 5.3.

5.2.3 The smart card only gives the pseudonym with the verifier to the terminal

This distribution will be interesting in case the terminal is owned by the verifier. Because the pseudonym with the verifier is already known to the verifier, this will not leak linking information.

However, if we look at the most complex function, i.e., VerifyCredOnNym, the pseudonym P_{UV} only appears in the proof of knowledge as the part $\mathrm{PK}\left\{ (\beta, \eta) : P_{UV}^2 = (a_V^2)^\beta (b_V^2)^\eta \bmod n_V \right\}$. Unfortunately, to execute this part of the proof of knowledge, the smart card is also involved as P_{UV} hides the user's master key. So this distribution gives no benefit compared to the distribution where the smart card keeps everything secret.

5.2.4 The smart card keeps everything secret

In the case that the smart card keeps everything secret and only uses the terminal as an interface, the workload is on the smart card.

If the terminal is managed by a third party and may gain totally no linking information, the smart card and organisation can communicate over an encrypted tunnel. This can be achieved by standard techniques which are beyond the scope of this thesis.

Modular exponentiations require by far the most calculation power. Therefore, we will use them as a measure for calculation costs. The following table gives the number of needed modular exponentiations per action in *idemix*:

action	number of exponentiations
FormNym	4
GrantCred	4
VerifyCred	10
VerifyCredOnNym	11

From the data on state-of-the-art smart card technology (see Appendix B) we see that one exponentiation takes about 174 milliseconds, thus the most costly operation VerifyCredOnNym would take approximately 2 seconds.

If we use a smart card which needs about 1.5 seconds to do one exponentiation, then VerifyCredOnNym takes approximately 17 seconds. In that case, we can not expect users to wait while a verification is done on the smart card.

5.2.5 Interesting distributions

In Table 5.1 it can be found which of the properties of *idemix* (as listed in Chapter 3) hold for each distribution described above.

Recapitulating this section, we have found two interesting distributions:

1. The smart card keeps only the master key secret.
2. The smart card keeps everything secret.

The second of these possibilities merely implements the whole user-side of *idemix* on a smart card. The first one will be worked out in the next section.

	the card gives all information to the terminal	the card only keeps the master key secret	the card only gives the verifier pseudonym	the card keeps everything secret
Unlinkable pseudonimity				
The organisation cannot link a pseudonym to an identity		x	x	x
The organisation cannot link two different pseudonyms to each other			x	x
The organisation cannot link two different uses of the same credential			x	x
Pseudonyms are bound to a user	x	x	x	x
Users cannot exchange credentials without giving other sensitive information	x	x	x	x
Unforgeable pseudonymous credential granting				
The organisation can grant credentials to users	x	x	x	x
The organisation only needs to know a user by pseudonym to grant a credential	x	x	x	x
The identity of a user remains secret to the organisation		x	x	x
All other pseudonyms of a user remain secret to the organisation			x	x
Zero-knowledge credential verification				
A user can show ownership of a pseudonym to the organisation				
... without revealing his identity		x	x	x
... without revealing other pseudonyms			x	x
A user can show ownership of a credential to the organisation				
... without revealing his identity		x	x	x
... without revealing other pseudonyms			x	x

Table 5.1: The properties of *idemix* under certain distributions if the organisation owns the terminal

5.3 The smart card only keeps the master key

In the case that the smart card only keeps the master key, but discloses the pseudonyms and credentials to the terminal, some unlinkability properties vanish. Table 5.2 shows what is disclosed with each action of *idemix*.

The terminal ...	The organisation ...
FormNym	
... learns the Pseudonym	... learns the Pseudonym
... does not learn the Master Key	... does not learn the Master Key
GrantCred	
... learns the Pseudonym	... learns the Pseudonym
... learns the Credential pair	... learns the Credential pair
... does not learn the Master Key	... does not learn the Master Key
VerifyCred	
... learns the Pseudonym	... learns the Pseudonym
... learns the Credential pair	... does not learn the Credential pair
... does not learn the Master Key	... does not learn the Master Key
VerifyCredOnNym	
... learns the Pseudonym with the Issuer	... does not learn the Pseudonym with the Issuer
... learns the Pseudonym with the Verifier	... learns the Pseudonym with the Verifier
... learns the Credential pair	... does not learn the Credential pair
... does not learn the Master Key	... does not learn the Master Key

Table 5.2: Disclosure of information if the smart card keeps only the user's master key secret

Of course, the right column of the table only holds if the organisation has no direct access to the terminal.

As we can see in the table, with GrantCred, VerifyCred and VerifyCredOnNym the terminal learns which pseudonym is coupled to a credential, and with VerifyCredOnNym the terminal can also link two pseudonyms to each other. The terminal never learns the user's master key.

To see what happens to the calculations, we have to distribute all calculations and data carefully over the smart card and terminal.

Distribution of calculations and data

Now we look at the functions FormNym, GrantCred, VerifyCred and VerifyCredOnNym.

FormNym

For pseudonym generation the protocol remains unchanged. The smart card chooses a random value s_{UO} and calculates $P_{UO} = a_O^{x_U} b_O^{s_{UO}} \bmod n_O$.

The user has to prove that he knows the master key x_{UO} and the hiding exponent s_{UO} which are bound to the shown pseudonym. This is done by a zero-knowledge proof of knowledge. Because all information that is needed to do so remains secret on the smart card, the terminal is used only as an interface for the smart card.

Therefore, the smart card has to do 2 exponentiations for the pseudonym generation and another 2 exponentiations for the proof of knowledge.

GrantCred

Again the zero-knowledge proof of knowledge requires 2 exponentiations to be done on the smart card.

Issuing a credential is done by the organisation. The terminal receives the credential triple (c_{UO}, e_{UO}, r_{UO}) and stores it or gives it to the smart card to let the card store this triple.

If the terminal can be trusted, verification of the credential triple by the user can be done on the terminal.

VerifyCred

When starting (and ending) this protocol, the smart card has the user's master key and the value s_{UO} which together form the pseudonym P_{UO}. The terminal knows the pseudonym P_{UO} and the credential triple (c_{UO}, e_{UO}, r_{UO}) such that

$$c_{UO}^{e_{UO}} = P_{UO}\, b_O^{r_{UO}}\, d_O \bmod n_O,$$

but does not know the values of x_U and s_{UO}. VerifyCred, as introduced in Section 3.4.5, is adapted to this situation as follows, where S is the Smart card, T is the Terminal and V is the Verifier:

1. T chooses $r_1, r_2 \in [0, 2^{2\ell_n})$, computes $A = c_{UO} h_O^{r_1} \bmod n_O$ and $B = g_O^{r_2} h_O^{r_1} \bmod n_O$ and sends the results to V.

2. T and S prove knowledge of the exponents in the pseudonym and knowledge of a signature by executing

$$\mathrm{PK}\left\{ (\alpha,\beta,\gamma,\delta,\epsilon,\zeta,\xi) : d_O^2 = (A^2)^\alpha (\frac{1}{a_O^2})^\beta (\frac{1}{b_O^2})^\gamma (\frac{1}{h_O^2})^\delta \wedge \right.$$
$$B^2 = (h_O^2)^\epsilon (g_O^2)^\zeta \wedge 1 = (B^2)^\alpha (\frac{1}{h_O^2})^\delta (\frac{1}{g_O^2})^\xi \wedge$$
$$\left. \beta \in [0, 2^{\ell_x}) \wedge \gamma \in [0, 2^{\ell_n}) \wedge \alpha \in [2^{\ell_e - 1}, 2^{\ell_e}) \right\}.$$

   Because the terminal cannot prove knowledge of β and γ, he passes this to the smart card. How this can be done for $\mathsf{PK}\{(\alpha,\beta,\gamma,\delta) : d_O^2 = (A^2)^\alpha (\frac{1}{a_O^2})^\beta (\frac{1}{b_O^2})^\gamma (\frac{1}{h_O^2})^\delta\}$ is shown in Figure 5.1.

   We will not work out the complete proof of knowledge in detail. But we describe what has to be altered in the protocol to achieve the desired distribution.

   In the commitment phase the values of r_α, r_δ, r_ϵ, r_ζ and r_ξ are chosen at random from the correct intervals by the terminal. However, the random values of r_β and r_γ are chosen by the smart card and the value $R_S = (1/a_O^2)^{r_\beta} (1/b_O^2)^{r_\gamma} \bmod n_O$ is given to the terminal. Then the terminal can calculate all the needed commitments and gives them to the verifier.
   The verifier replies by giving a challenge.
   The responses s_α, s_δ, s_ϵ, s_ζ and s_ξ are calculated by the terminal. But again, the responses for s_β and s_γ have to be calculated by the smart card and given to the terminal. Then the terminal has all the values needed to convince the verifier that the user knows all values of which knowledge had to be proven.

We see that in the commitment phase of the proof of knowledge, the smart card has to do 2 exponentiations.

It can easily be seen that the zero-knowledge proofs of knowledge remain secure under this distribution, because the game between the terminal and the smart card can be seen as a stand-alone zero-knowledge proof of knowledge.

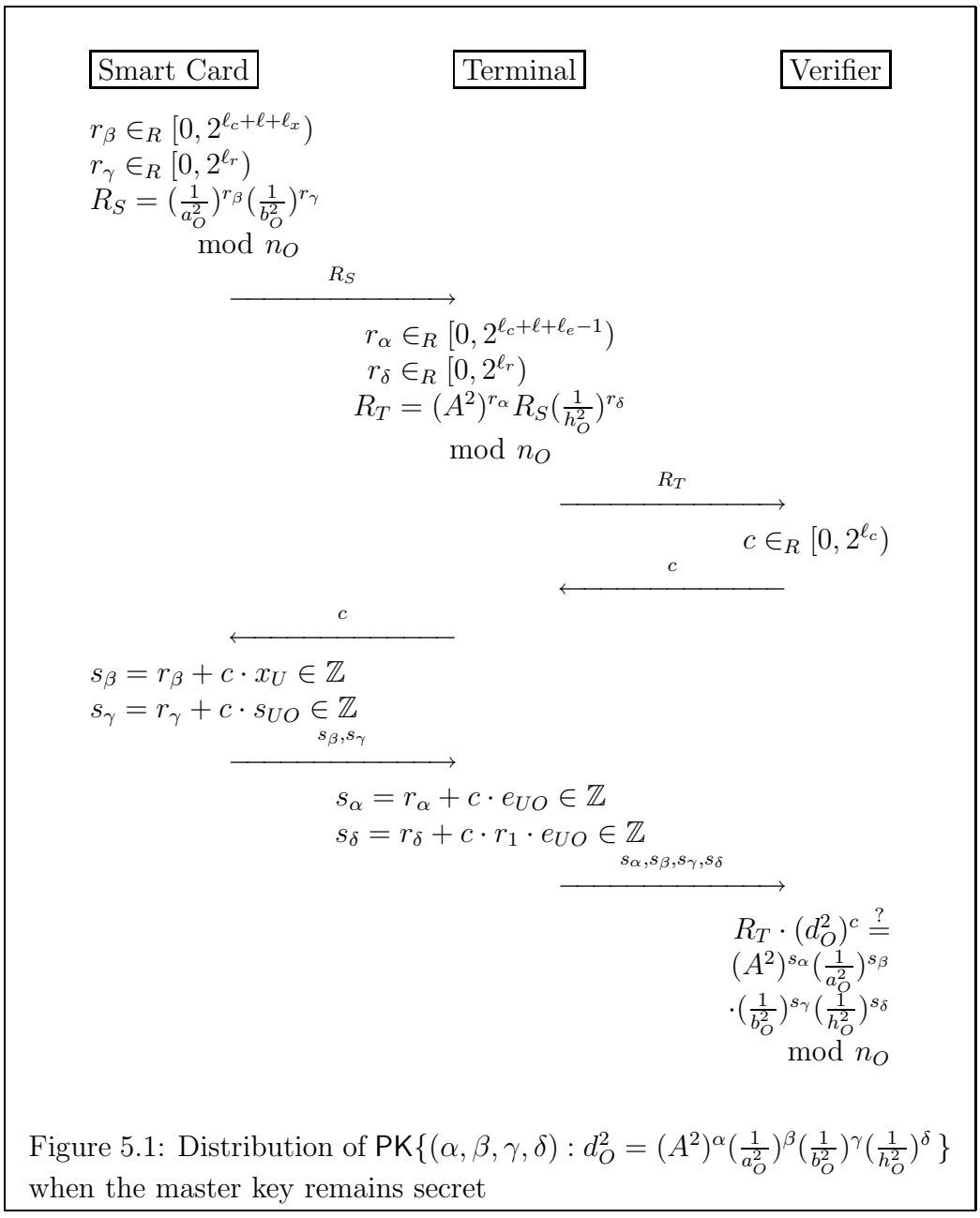

Figure 5.1: Distribution of $\mathsf{PK}\{(\alpha, \beta, \gamma, \delta) : d_O^2 = (A^2)^\alpha (\frac{1}{a_O^2})^\beta (\frac{1}{b_O^2})^\gamma (\frac{1}{h_O^2})^\delta\}$ when the master key remains secret

VerifyCredOnNym

When starting (and ending) this protocol, the smart card has the user's master key x_U, the value s_{UI} that together with x_U forms the pseudonym P_{UI} and the value s_{UV} that together with x_U forms the pseudonym P_{UV}. The terminal knows the pseudonyms P_{UI}, P_{UV} and the credential triple (c_{UI}, e_{UI}, r_{UI}) such that

$$c_{UI}^{e_{UI}} = P_{UI}\, b_I^{r_{UI}}\, d_I \bmod n_I,$$

but does not know the values of x_U, s_{UI} and s_{UV}. VerifyCredOnNym is adapted to this situation as follows, where S is the Smart card, T is the Terminal and V is the Verifier:

1. T chooses $r_1, r_2 \in [0, 2^{2\ell_n})$, computes $A = c_{UI} h_I^{r_1} \bmod n_I$ and $B = g_I^{r_2} h_I^{r_1} \bmod n_I$ and sends the results to V.

2. T and S prove knowledge of the exponents in the pseudonym and knowledge of a signature by executing

$$\begin{aligned} \mathrm{PK}\Big\{ (\alpha,\beta,\gamma,\delta,\epsilon,\zeta,\xi,\eta) : d_I^2 = (A^2)^{\alpha}(\frac{1}{a_I^2})^{\beta}(\frac{1}{b_I^2})^{\gamma}(\frac{1}{h_I^2})^{\delta} \wedge \\ B^2 = (h_I^2)^{\epsilon}(g_I^2)^{\zeta} \wedge 1 = (B^2)^{\alpha}(\frac{1}{h_I^2})^{\delta}(\frac{1}{g_I^2})^{\xi} \wedge \\ P_{UV}^2 = (a_V^2)^{\beta}(b_V^2)^{\eta} \bmod n_V \wedge \\ \beta \in [0, 2^{\ell_x}) \wedge \gamma \in [0, 2^{\ell_n}) \wedge \alpha \in [2^{\ell_e - 1}, 2^{\ell_e}) \Big\}. \end{aligned}$$

   Because the terminal cannot prove knowledge of β, γ and η, he passes this to the smart card. Therefore, in the commitment phase the values of r_α, r_δ, r_ϵ, r_ζ and r_ξ are chosen at random from the correct intervals by the terminal. However, the random values of r_β, r_γ and r_η are chosen by the smart card and the values $R_{S,P_{UI}} = (1/a_I^2)^{r_\beta}(1/b_I^2)^{r_\gamma} \bmod n_I$ and $R_{S,P_{UV}} = (1/a_V^2)^{r_\beta}(1/b_V^2)^{r_\gamma} \bmod n_I$ are given to the terminal. Then the terminal can calculate all the needed commitments and gives them to the verifier.
   The verifier replies by giving a challenge.
   The responses s_α, s_δ, s_ϵ, s_ζ and s_ξ are calculated by the terminal. But again, the responses s_β, s_γ and s_η have to be calculated by the smart card and given to the terminal. Then the terminal has all the values needed to convince the verifier that the user knows all values of which knowledge had to be proven.

We see that in the commitment phase of the proof of knowledge, the smart card has to do 4 exponentiations.

When using this distribution, the following number of exponentiations has to be done on the smart card, assuming that the terminal can be trusted:

action	number of exponentiations
FormNym	4
GrantCred	2
VerifyCred	2
VerifyCredOnNym	4

The most expensive functions FormNym and VerifyCredOnNym now only take $4 \cdot 174$ ms ≈ 0.7 seconds. If we use a smart card that needs about 1.5 seconds per exponentiation, these functions each take about 6 seconds.

5.4 Analysis and preferences

To investigate which distribution is preferred, we first have to distinguish between three different possible terminal owners: the user, the organisation or a third party.

A terminal owned by **the user** can be a digital wallet, for example, a cell phone equipped with a SIM-card containing the user's master key. Another possibility is a card interface on the user's personal computer.

In this situation we can expect that the terminal will not leak linking information to organisations. Therefore the smart card can give the needed pseudonyms and credentials to the terminal. The best solution is to do this just before the pseudonyms and credentials are needed, and let the terminal delete them right afterwards. To protect the user's master key, the smart card keeps this key secret.

If **the organisation** (issuer and/or verifier) owns the terminal, then for the user there is no difference between the terminal and the organisation. In that case some properties of *idemix* vanish. Table 5.1 gives an overview of the remaining properties of *idemix* for the various distributions.

To preserve the unlinkability of pseudonyms in *idemix*, we must choose to keep everything secret on the smart card. It is not possible to gain some benefits on calculation time by giving some calculations to the terminal, without giving non-blinded secret values.

A possibility which we have not discussed, is to blind the secret values x_U, s_{UO}, P_{UI} and/or the credential triple (c_{UO}, e_{UO}, r_{UO}) and perform calculations with these blinded values on the terminal. Whether we can efficiently and securely outsource calculations in this way remains a questions for further research.

It also is possible that the terminals are provided by **a third party** and are used to let the smart card communicate with the organisation. In that case the user cannot expect the terminal to be secure. So the smart card has to do all the calculations. In order to communicate securily, the user and the organisation can encrypt their communication when it is transported over this terminal.

If the user trusts the terminal provider to hide his linking information, the user can choose to keep his master key on the smart card and give other needed information to the terminal.

Chapter 6

Conclusions

In the previous chapter we have investigated in which way *idemix* can be implemented in a system with a smart card and a terminal. This section gives a small summary of the progress we made on researching this question. We also make some conclusions on the results of this research. Moreover, we will list some suggestions for further research.

6.1 Conclusions

While studying the properties and protocols of *idemix* it became clear that it is necessary for security to choose all values from the correct intervals and thus integrating a smart card in *idemix* has to be done with a lot of care.

As described in Section 4.6.3, we found that there are no exact interval proofs needed in the standard protocols of *idemix*. This saves a lot of calculation power.

If the terminal is provided by the organisation, we unfortunately have to conclude that it is not possible to reduce the weight of calculations that have to be done by the smart card without revealing linking information.

The function VerifyCredOnNym takes about 2 seconds on the quickest smart card. If we use a smart card which needs about 1.5 seconds to do one exponentation, then VerifyCredOnNym takes approximately 17 seconds.

On the other hand, if the user owns the terminal, we can choose to give the pseudonyms and credentials that are needed in a transaction to the terminal and leave the necessary calculations to the terminal as well.

If only the user's master key remains secret on the terminal, for the functions FormNym and VerifyCredOnNym only 4 exponentiations have to be done on the smartcard. With the quickest smart card from Appendix B it takes no more than a second. With other mentioned smart cards it will take more or less 6 seconds. Because the needed exponentiations only indicate something about the processing times, we have to mention here that we don't know exactly how an implementation on the smart card affects this.

We may conclude that *idemix* can be implemented in a system which makes use of a smart card and a terminal, where the user's master key is contained on and handled by the card.

6.2 Further research

While doing this research, some questions arose which could not be treated in this thesis. Therefore we mention here some suggestions for further research.

First we notice that for the signature scheme given by Lysyanskaya (Section 3.2) it is necessary that the hiding exponent of the commitment is chosen uniformly at random. The proof of security makes use of this fact. We have neither found a different proof of security which would make the requirement of randomness superfluous, nor any argument that shows that it is not secure to use non-random hiding exponents. If it were possible to safely use non-random hiding exponents, the exponent r_{UO} in the credential triple would be superfluous.

We also found that combining the zero-knowledge proofs of knowledge building blocks is a complicated job that has to be done very precisely. Endre Bangerter, who has done his PhD with Jan Camenisch, already states in a slide show [Ban07] that it would be useful to develop an automatic protocol generator for zero-knowledge proofs of knowledge. A complicating factor is that the resulting protocols must remain secure when combining the basic blocks.

In the previous chapter, we saw that it is impossible to give information to the terminal if that terminal is owned by the organisation, without leaking linking information. However, it is not clear yet whether we can move some calculation load to the terminal. It may be possible that the terminal can do some calculations in an oblivious way by hiding the input or output. Whether or not it is possible to securely outsource computations is a question for further research.

Bibliography

[Ban07] E. Bangerter. *Zero-knowledge proofs of knowledge - a journey from the foundations of theoretical computer science to security engineering.* Slides for a seminar of the Computer Science Division of the Berner Fachhochschule, February 2007. `https://www2.ti.bfh.ch/fbi/Colloques`.

[Bou00] F. Boudot. *Efficient proofs that a committed number lies in an interval.* In *Advances in Cryptology – EUROCRYPT 2000*, vol. 1807 of *Lecture Notes in Computer Science*, pp. 437–450. Springer, 2000.

[Bra00] S. A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy.* MIT Press, Cambridge, MA, USA, 2000.

[Cam05] K. Cameron. *The laws of identity.* A whitepaper found on Kim Cameron's Identity Weblog, `http://www.identityblog.com/?p=354`, May 2005.

[Cha85] D. Chaum. *Security without identification: transaction systems to make big brother obsolete.* Communications of the ACM, 28 (1985), no. 10:1030–1044.

[CL01a] J. Camenisch and A. Lysyanskaya. *An efficient system for non-transferable anonymous credentials with optional anonymity revocation.* Cryptology ePrint Archive, Report 2001/019, March 2001. Extended version of [CL01b], published on `http://eprint.iacr.org/2001/019`.

[CL01b] J. Camenisch and A. Lysyanskaya. *An efficient system for non-transferable anonymous credentials with optional anonymity revocation.* In *Advances in Cryptology – EUROCRYPT 2001*, vol. 2045 of *Lecture Notes in Computer Science*, pp. 93–118. Springer, 2001.

[CL02] J. Camenisch and A. Lysyanskaya. *A signature scheme with efficient protocols.* In *International Conference on Security in Communication Networks – SCN*, vol. 2576 of *Lecture Notes in Computer Science*, pp. 268–289. Springer, 2002.

[CS97] J. L. Camenisch and M. A. Stadler. *Efficient group signature schemes for large groups.* In B. Kaliski (ed.), *Advances in Cryptology – CRYPTO '97*, vol. 1294 of *Lecture Notes in Computer Science*, pp. 410–424. Springer, 1997.

[Dam00] I. Damgård. *Efficient concurrent zero-knowledge in the auxiliary string model.* In *Advances in Cryptology – EUROCRYPT 2000*, vol. 1807 of *Lecture Notes in Computer Science*, pp. 418–430. Springer, 2000.

[DF02] I. Damgård and E. Fujisaki. *A statistically-hiding integer commitment scheme based on groups with hidden order.* In *Advances in Cryptology – ASIACRYPT 2002*, vol. 2501 of *Lecture Notes in Computer Science*, pp. 77–85. Springer, 2002.

[DN07] I. Damgård and J. B. Nielsen. *Commitment schemes and zero-knowledge protocols (2007).* An introduction, `http://www.daimi.au.dk/~ivan/ComZK07.pdf`, February 2007.

[FO97] E. Fujisaki and T. Okamoto. *Statistical zero knowledge protocols to prove modular polynomial relations.* In *Advances in Cryptology - CRYPTO '97*, vol. 1294 of *Lecture Notes in Computer Science*, pp. 16–30. Springer, 1997.

[Gol04] O. Goldreich. *Zero-knowledge twenty years after its invention.* A tutorial, `http://www.wisdom.weizmann.ac.il/~oded/zk-tut02.html`, March 2004.

[Lys02] A. Lysyanskaya. *Signature Schemes and Applications to Cryptographic Protocol Design.* Ph.D. thesis, Massachusetts Institute of Technology, September 2002.

[MOV96] A. J. Menezes, P. C. V. Oorschot and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1996.

[Ped92] T. P. Pedersen. *Non-interactive and information-theoretic secure verifiable secret sharing.* In *Advances in Cryptology -*

CRYPTO '91, vol. 576 of *Lecture Notes in Computer Science*, pp. 129–140. Springer, 1992.

[QGB90] J.-J. Quisquater, L. C. Guillou and T. A. Berson. *How to explain zero-knowledge protocols to your children.* In *Advances in Cryptology - CRYPTO '89*, vol. 435 of *Lecture Notes in Computer Science*, pp. 628–631. Springer, 1990.

[Sch89] C. P. Schnorr. *Efficient identification and signatures for smart cards.* In *Advances in Cryptology - CRYPTO '89*, vol. 435 of *Lecture Notes in Computer Science*, pp. 239–252. Springer, 1989.

[Sch96] B. Schneier. *Applied Cryptography.* John Wiley & Sons, 2nd edn., 1996.

Appendix A

Calculating square roots of large numbers

In the interval proof of Boudot the calculation of the floor of a square root is used. To calculate a square root with high precision a lot of memory and calculation power is needed, but luckily we only need the integer part. We will see that using Newton's method, this can be done efficiently.

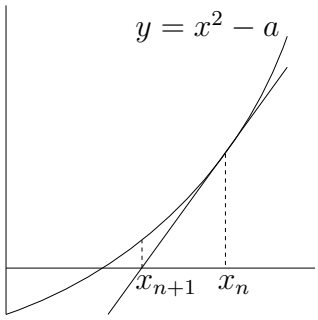

Figure A.1: Illustration of Newton's method for finding a root of $x^2 - a$.

Newton's method is an algorithm for finding approximations $x_0, x_1, x_2, \ldots$ to a zero of a real-valued function f. It uses slopes of a tangent at points of the graph of f. In each iteration step n, we take as the next approximation x_{n+1} the zero of the tangent of $f(x)$ at $x = x_n$. This can be written as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Although there are functions and initial values x_0 for which this method doesn't converge, in the case of finding square roots this method will lead to a good approximation in a few steps.

For finding a square root of a, we look for a root of $x^2 - a = 0$. In this case, the iteration takes the form

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{x_n}{2} + \frac{a}{2x_n}.$$

Because we are looking for the floor of the square root, we can take the floor of $\frac{x_n}{2} + \frac{a}{2x_n}$ in each iteration step. The iteration is finished when the outcomes don't change anymore.

Analysis

The choice of the starting value is important, choosing a starting value close to the square root we are looking for gives (almost) quadratic convergence. However, choosing a value far away from the square root gives a linear convergence.

This can be seen as follows: First, we examine the case that we start close to the square root:

$$\begin{aligned}
x_0 &= (1+\epsilon)\sqrt{a} \\
x_1 &= \frac{x_0}{2} + \frac{a}{2x_0} \\
&= \left(\frac{1+\epsilon}{2} + \frac{1}{2(1+\epsilon)}\right)\sqrt{a} \\
&= \left(\frac{2+2\epsilon+\epsilon^2}{2+2\epsilon}\right)\sqrt{a} \\
&= \left(1 + \frac{\epsilon^2}{2+2\epsilon}\right)\sqrt{a} \\
&\approx \left(1 + \frac{\epsilon^2}{2}\right)\sqrt{a}.
\end{aligned}$$

Hence close to the square root, the convergence is quadratic in the distance factor.

Now we examine the case that we start with a much larger value than the square root:

$$\begin{aligned} x_0 &= 2^k\sqrt{a} \text{ with } k \gg 0 \\ x_1 &= \frac{x_0}{2} + \frac{a}{2x_0} \\ &= \left(2^{k-1} + 2^{-k-1}\right)\sqrt{a} \\ &\approx \left(2^{k-1}\right)\sqrt{a}, \text{ so using induction} \\ x_i &\approx \left(2^{k-i}\right)\sqrt{a} \text{ as long as } k \gg i. \end{aligned}$$

For a positive starting value much smaller than the square root we see

$$\begin{aligned} x_0 &= 2^{-k}\sqrt{a} \text{ with } k \gg 0 \\ x_1 &= \frac{x_0}{2} + \frac{a}{2x_0} \\ &= \left(2^{-k-1} + 2^{k-1}\right)\sqrt{a} \\ &\approx \left(2^{k-1}\right)\sqrt{a} \end{aligned}$$

and we are in the same situation as for a value much larger than the square root.

We conclude that it is efficient to choose a starting value near to the end value, and therefore we choose as the starting value a number with half of the bitlength as the square has, i.e.,

$$x_0 = 2^{\lfloor 1/2 \log a \rfloor}.$$

Experiments

Experiments in Mathematica confirm this analysis. For a n-bit square it takes approximately $n/2 + 2^{\lfloor 1/2 \log a \rfloor} - 1$ steps if chosen 1 as the starting value, while it only takes approximately $\log n$ steps if chosen $2^{\lfloor 1/2 \log a \rfloor}$ as the starting value.

The output of Mathematica is as follows:

```
In[1]:= bitlength = 2^8;

In[2]:= NewtonSqrt[square_, bitlength_] := (
          xn = 0; xnplus1 = 2^Floor[bitlength / 2]; step = 0;
          While[xn != xnplus1,
           xn = xnplus1;
           step++;
           Print["step ", step, " approximation: ", xn];
           xnplus1 = Floor[(xn / 2) + (square / (2 * xn))]];
          xn)

In[3]:= kwadraat = Random[Integer, {0, 2^bitlength - 1}]

Out[3]= 10846990975321878678778555708522144954360064303817410579095063235680800867495O

In[4]:= wortel = NewtonSqrt[kwadraat, bitlength]

        step 1 approximation: 340282366920938463463374607431768211456

        step 2 approximation: 329523391146859269186018050648028689422

        step 3 approximation: 329347750262452926672607764603574045507

        step 4 approximation: 329347703427883310729530942110137530106

        step 5 approximation: 329347703427879980697336576254727124771

        step 6 approximation: 329347703427879980697336576237892159322

Out[4]= 329347703427879980697336576237892159322
```

Appendix B

Modular arithmetic on a smart card

In order to know how much calculation power is needed for the actions in *idemix*, we need to understand some methods to do binary calculations. In this appendix we survey binary addition, substraction, multiplication, division and exponentiation. Exponentiation can be optimised in different ways and we look at some of this methods to approximate the time a smart card needs to do such an exponentiation.

B.1 Addition, subtraction and multiplication

Binary addition is done similar to decimal addition, namely as follows: Write the numbers to add under each other and right align them. For each position from the right to the left: if both are zero, then the outcome is zero, if one digit is a zero and one digit is a one, then the outcome is one and if both are one, then the outcome is zero and we put a 1 in the carry. If we had also something in the carry, a zero becomes a one, and a one becomes a zero with 1 in the carry. Do this for every position.

Example:

$$\begin{array}{cccc} & & {\scriptstyle 1} & \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ \hline 1 & 1 & 1 & 0 \end{array} +$$

Binary subtraction is almost done in the same way: write the numbers to subtract under each other and right align them. For each position from the right to the left: if both are zero or both are one, then the outcome is zero, if only the upper digit is a one, then the outcome is one, if only the lower digit is a one, then the outcome is also one, but there is put a one in the carry to subtract later. If we had also something in the carry, a one becomes zero, a zero becomes a one, with one in the carry.

Example:

$$\begin{array}{cccc} {\scriptstyle 1} & & & \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 \end{array} \; -$$

Binary multiplication can be done as follows: write the numbers to multiply under each other and right align them. For each position of the second number from the right to the left: if the digit is a one, put the upper number padded with a number of zero's on the right as much as the position of the lower digit under each other. Finally add all this numbers by binary addition.

Example:

$$\begin{array}{cccccc} & & 1 & 0 & 0 & 1 \\ & & 0 & 1 & 0 & 1 \\ \hline & & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ \hline 1 & 0 & 1 & 1 & 0 & 1 \end{array} \; \begin{array}{c} \\ * \\ \\ + \\ \end{array}$$

Modular binary addition and **subtraction** can easily be done by the following rules:

$$a + b \mod n = \begin{cases} a + b & \text{if } a + b < n \\ a + b - n & \text{if } a + b \geq n \end{cases}$$

$$a - b \mod n = \begin{cases} a - b & \text{if } a - b \geq 0 \\ a - b + n & \text{if } a - b < 0 \end{cases}$$

Modular binary multiplication is done by binary multiplication and then calculating the reminder of dividing by n.

B.2 Multiplicative inverses

Inverses in $\mathbb{Z}_n$ can be computed using the extended Euclidian algorithm. This algorithm can be easily found in the mathematical literature, for example in [MOV96].

Assume we want to calculate a^{-1} for some $a \in \mathbb{Z}_n$. If and only if $\gcd(a, n) = 1$, the inverse $a^{-1} \bmod n$ does exist. Use the extended Euclidian algorithm to find integers x and y such that $ax + ny = \gcd(a, n) = 1$, then $ax = 1 \bmod n$, thus a^{-1} is given by x.

B.3 Modular exponentiation

Repeated square-and-multiply
A simple and efficient method for modular binary exponentiation is called *repeated square-and-multiply*. To calculate b^e, observate that e can be written as

$$e = \sum_{i=0}^{k} e_i 2^i,$$

with $e_i \in 0, 1$ for all i, from which follows that

$$b^e = \prod_{i=0}^{k} (b^{2^i})^{e_i}.$$

Observing this, we construct the next algorithm:

1. Set $result = 1$ and $B = b$.
2. For i from 0 to k do the following:
   2.1 If $e_i = 1$ then set $result = B \cdot result \bmod n$.
   2.2 Set $B = B^2 \bmod n$.
3. Return *result*.

Chinese Remainder Theorem
Normally used on smart cards is the Chinese Remainder Theorem (CRT) to calculate modular exponentiations, especially signing with RSA. Using CRT is only possible if the factors of the modulus are known, which is not the case in our situation. Therefore using CRT is not possible.

B.4 Complexity of modular operations

The following table (also found in [MOV96]) shows the bit complexity of the mentioned modular operations:

Operation		Bit complexity
Modular addition	$a + b \bmod n$	$O(\log n)$
Modular subtraction	$a - b \bmod n$	$O(\log n)$
Modular multiplication	$a \cdot b \bmod n$	$O((\log n)^2)$
Modular inversion	$a^{-1} \bmod n$	$O((\log n)^2)$
Modular exponentiation	$a^k \bmod n, k < n$	$O((\log n)^3)$

Table B.1: Bit complexity of modular operations

As we see, modular exponentiation is the most expensive operation by far. To be more concrete: if we expect half of bits in the exponent to be 1, we have to calculate $\log n$ squares and do $1/2 \log n$ multiplications. For a 2048 bit modulus, an exponentiation will take approximately 3072 multiplications.

While analysing the mathematical operations in *idemix*, we have to concentrate on exponentiations.

B.5 State-of-the-art smart cards

To see what is the state-of-the-art in smart card technology, we give an overview of smart card specifications of some well known smart card manufacturers.

Infineon	SLE 88P family	
Clock speed		66 MHz
Memory	RAM	32 Kbyte
	EEPROM / Flash	1 Mbyte
RSA signing	1024 CRT	14 ms
	2048 CRT	58 ms
NXP (Philips)	SmartMX P5CN144/P5CC144/P5CD144	
Clock speed		30 MHz
Memory	ROM	264 Kbyte
	RAM	6 Kbyte
	EEPROM	144 Kbyte
RSA signing	1024 CRT	99 ms

Gemalto	GemsafeXpresso 64K (April '07)	
Type		Java Card™
Memory	EEPROM / Flash	64 Kbyte
RSA signing	1024	250 ms
	2048	1400 ms
STMicroelectronics	ST22N256(-A) (June '06)	
Card type		Java Card™
Clock speed		33 MHz
Memory	ROM	394 Kbyte
	RAM	16 Kbyte
	EEPROM / Flash	256 Kbyte
RSA signing	1024 CRT	79 ms
	1024 (w/o CRT)	242 ms
	2048 CRT	485 ms
	2048 (w/o CRT)	1700 ms

Table B.2: Overview of smart card specifications

Because we can not use the Chinese Remainder Theorem, we have to multiply the CRT times roughly by 3 to obtain the real exponentiation times needed by these processors. If the exponentiation times are given for 1024-bit exponents, we have to multiply it by 4 to obtain the times for 2048-bit exponentations. That gives us the following times per manufacturer for a 2048-bit exponentiation:

Manufacturer	Exponentiation time
Infineon	174 ms
NXP	1188 ms
Gemalto	1400 ms
STMicroelectronics	1700 ms

We see that the Infineon card is the fastest with approximately 174 ms for an ordinary 2048-bit exponentiation.

Appendix C

List of symbols

symbol	value	meaning
$\log$	$\log_2$	binary logarithm
$\in_R$		(uniformly) random integer
$[a, b]$		interval in $\mathbb{Z}$, including the endpoints
(a, b)		interval in $\mathbb{Z}$, excluding the endpoints
$\mathbb{Z}_n$		integers modulo n
$\mathbb{Z}_n^*$		group of units in $\mathbb{Z}_n$
$\phi(n)$		Euler totient function
QR_n		group of quadratic residues in $\mathbb{Z}_n^*$
U		the User
O		the Organisation (can be I or V)
I		the Issuer
V		the Verifier
PK_O	(n_O, a_O, b_O, d_O)	public key of the Organisation
SK_O	(p_O, q_O)	secret key of the Organisation
(c_{UO}, e_{UO}, r_{UO})	see 3.4.4	credential for the User at O
x_U		User's master key
s_{UO}		hiding exponent in P_{UO}
P_{UO}	$a_O^{x_U} b_O^{s_{UO}} \bmod n_O$	pseudonym of the User at O
ℓ_n	see 3.5	bitlength of the modulus n
ℓ_x	see 3.5	bitlength of the user's master key x_U
ℓ	see 3.5	a security parameter
ℓ_s	$\ell_n + \ell_x + \ell$	bitlength of the hiding exponent s_{UO}
ℓ_x	see 3.5	bitlength of the user's master key x_U
ℓ_c	see 3.5	bitlength of the challenge
ℓ_e	$\ell_x + 2$	bitlength of the signature exponent e
$\mathrm{PK}\left\{ (\alpha, \dots) : \dots \right\}$		zero-knowledge proof of knowledge
$\stackrel{?}{=}$		equality test
ℓ_f	$\ell_s + 2\ell_c + \ell - t + 1$	see 4.6.1