# Efficient Implementations of Attribute-based Credentials on Smart Cards

**Pim Vullers**

Copyright © 2014 Pim Vullers

IPA Dissertation Series: 2014-15
ISBN: 978-94-6259-376-3
NUR: 980

Typeset using $\LaTeX\,2_\varepsilon$ (TEXLive 2013)
Cover design by Koen Vullers
Printed by Ipskamp Drukkers (`http://proefschriften.net/`)

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). The author was employed at the Radboud University Nijmegen.

This research was sponsored by Trans Link Systems / Open Ticketing.

# Efficient Implementations of Attribute-based Credentials on Smart Cards

Proefschrift

ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. dr. Th.L.M. Engelen,
volgens besluit van het college van decanen
in het openbaar te verdedigen op **vrijdag 28 november 2014**
om **10.30 uur** precies.

door

Pim Vullers

geboren op 20 juni 1986
te Venlo

# Samenvatting

We leven in een wereld waarin computers een steeds grotere rol spelen. Daarom wordt het alsmaar belangrijker om gebruikers en objecten digitaal te kunnen identificeren. Om dit te bereiken gebruiken veel bestaande systemen unieke nummers, denk bijvoorbeeld aan je burgerservicenummer (BSN). Dit is een eenvoudige oplossing, maar dit maakt het ook gemakkelijk om iemands handelingen te volgen. Een privacyvriendelijker alternatief is om attributen te gebruiken voor authenticatie en autorisatie in de digitale wereld. Deze attributen, of eigenschappen, kunnen gecombineerd worden in een soort cryptografische container die dan gecertificeerd wordt door een bevoegde autoriteit. Met behulp van deze attributen kan een gebruiker dan geauthentiseerd worden, om toegang te krijgen tot bepaalde systemen of om gebruik te maken van een dienst, door enkel de relevante gebruikerseigenschappen te delen.

In dit proefschrift bespreken we drie privacyvriendelijke technologieën die authenticatie op basis van attributen mogelijk maken. Deze technologieën, waarvoor we efficiënte implementaties voor op een chipkaart hebben ontwikkeld, zijn:

**Self-blindable Credentials** Deze technologie maakt gebruik van cryptografie op basis van elliptische krommen. Het systeem laat de meeste berekeningen door de kaartlezer uitvoeren, wat een zeer compacte implementatie op de chipkaart mogelijk maakt. Helaas is de ondersteuning voor elliptische kromme cryptografie op chipkaarten beperkt tot de standaard algoritmen. Dit maakt de ontwikkeling van varianten op deze technologie lastig. In vergelijking met de andere technologieën bieden de Self-blindable Credentials slechts beperkte mogelijkheden.

**U-Prove** De uitgifte- en verificatieprotocollen van U-Prove zijn gebaseerd op de methode voor geblindeerde handtekeningen van Schnorr en zijn zero-knowledge proofs (bewijzen waarbij geen kennis overgedragen wordt). Deze technologie biedt op dit moment de snelste implementatie voor verificatie met attributen. Met betrekking tot privacy is er echter een groot probleem: U-Prove beschermt niet tegen het aan elkaar koppelen van meerdere verificatiesessies. Dit betekent dat de verificatiegegevens functioneren als een pseudoniem voor de gebruiker.

**Identity Mixer** Deze technologie is gebaseerd op het werk van Camenisch en Lysyanskaya dat een methode voor digitale handtekeningen definieert. Deze methode bevat protocollen voor geblindeerde handtekeningen en zero-knowledge proofs, die gebruikt worden voor de uitgifte en verificatie van de attributen. In vergelijking met de andere implementaties behalen we hier niet de beste prestaties, maar deze

technologie biedt wel uitgebreide mogelijkheden en geeft een goede bescherming. Zo kunnen de attributen bijvoorbeeld meerdere keren gebruikt worden zonder dat de verschillende verificatiesessies traceerbaar worden.

Het doel van het onderzoek in dit proefschrift was om efficiënte chipkaart implementaties te ontwikkelen voor op attributen gebaseerde authenticatie en het vergelijken van de verschillende technologieën. Dit heeft geresulteerd in een gedetailleerde beschrijving en bespreking van de bovengenoemde technologieën en de bijbehorende chipkaart implementaties[1]. Deze implementaties bieden, ten tijden van het schrijven van dit proefschrift, de beste prestaties voor attributen op een chipkaart.

Daarnaast heeft de succesvolle ontwikkeling van deze chipkaart implementaties de basis gelegd voor het IRMA project. Dit is een lopend onderzoeks- en ontwikkelingsproject dat zich richt op authenticatie op basis van attributen en de toepassing daarvan in de praktijk. Voor meer informatie over het IRMA project, zie `https://www.irmacard.org/`.

[1]De broncode van deze implementaties is beschikbaar op `https://github.com/pimvullers/`.

# Voorwoord

Het is al weer vijf jaar geleden dat ik in Nijmegen ben gestart met mijn promotietraject. Aan de ene kant is het een traject dat je zelfstandig moet afleggen, maar aan de andere kant is het ook zeker een traject dat je niet alleen kunt doorlopen. Daarom wil ik hierbij iedereen bedanken die me tijdens deze periode geholpen heeft. Ik wil niemand te kort doen, maar toch wil ik een aantal personen in het bijzonder noemen.

Om te beginnen mijn promotor, Bart, die tevens de taak van dagelijks begeleider op zich genomen heeft. Niet alleen heb je me ondersteund bij het uitvoeren van mijn onderzoek en het schrijven van mijn artikelen, maar ook met het promoten van mijn resultaten en het opdoen van ervaring binnen het onderwijs. Ondanks je drukke agenda was er altijd wel tijd om mijn dagelijkse beslommeringen te bespreken. Bart, bedankt.

Ook de artikelen waarop dit proefschrift gebaseerd is heb ik niet alleen geschreven. Ik wil dan ook mijn collega's, Wojciech, Gergely, Bart, Jaap-Henk en Lejla van harte bedanken voor hun hulp. Wojciech en Erik wil ik ook graag bedanken voor het delen van hun kennis en ervaring met het programmeren van smartcards. Hier heb ik veel van geleerd en dit heb ik in dit werk veelvuldig toegepast.

Daarnaast wil ik ook mijn medepromovendi bedanken voor de leuke tijd die ik gehad heb. Bij onderzoeksschool IPA was het altijd gezellig, in het bijzonder het gezelschap van Carst en Jeroen, mijn medestudenten uit mijn studietijd in Eindhoven, en Wouter en Gergely, mijn collega's uit Nijmegen, en natuurlijk de organisatie van IPA, Meivan en Tim. Ook heb ik mijn volledige promotie op een ruime kamer gezwoegd met vele andere promovendi. Uit de gezamenlijke arbeid heb ik veel energie geput, maar zeker ook uit onze gesprekken en werkonderbrekingen. In het bijzonder staan de kleine security studie projectjes samen met Gerhard en Joeri me nog goed bij.

Tijdens mijn promotie heb ik ook de mogelijkheid gehad om bij Microsoft stage te lopen. Zo heb ik ervaring op kunnen doen en in de keuken kunnen kijken bij een onderzoeksafdeling bij een groot bedrijf. Christian, bedankt voor het regelwerk om de stage mogelijk te maken en je begeleiding tijdens mijn tijd bij Microsoft.

Uit mijn onderzoek is het IRMA project voortgekomen dat verder onderzoek doet naar het gebruik van attributen in de praktijk. Er is veel werk in gaan zitten, en ik ben blij dat ik dit niet alleen heb hoeven doen. Wouter, Gergely, Ronny, Roland, Maarten, Jaap-Henk, Antonio, Bart en Martijn, bedankt.

Ook wil ik mijn commissieleden bedanken voor de tijd die ze gestoken hebben in het beoordelen van mijn proefschrift. In het bijzonder wil ik Sjouke bedanken, omdat je ook al mijn begeleider bent geweest tijdens mijn afstudeerproject in Luxemburg.

Natuurlijk heeft al dit werk ook zijn invloed gehad op mijn persoonlijke leven. Tijdens mijn promotie zullen mijn broers Bas en Koen als mijn paranimfen optreden. Ook heeft Koen de tijd genomen om een omslag te ontwerpen voor dit proefschrift, en me daar veel werk mee uit handen genomen. Van mijn ouders, Jan en Gerrie, heb ik altijd de steun gehad voor deze promotie, ook al was het niet altijd te begrijpen waar ik dan precies mee bezig was.

Als laatste wil ik Marlou bedanken, die het niet altijd even eenvoudig heeft gehad tijdens mijn promotie (in het bijzonder tijdens het schrijven van dit proefschrift), maar me toch altijd ondersteund heeft om dit werk af te ronden. Bedankt.

*Pim Vullers*
*Nooitgedacht, oktober 2014*

# Efficient Implementations of Attribute-based Credentials on Smart Cards

Doctoral Thesis

to obtain the degree of doctor
from Radboud University Nijmegen
on the authority of the Rector Magnificus, prof. dr. Th.L.M. Engelen,
according to the decision of the Council of Deans
to be defended in public on **Friday, November 28, 2014**
at **10.30 hours**.

by

Pim Vullers

Born on June 20, 1986
in Venlo, The Netherlands

# Abstract

In a world where computers are involved in most aspects of our lives, it becomes more and more important to digitally identify entities. To achieve this goal, many existing systems use unique identifiers. This is a simple solution, but also makes it easy to trace the user's actions. A privacy-friendly alternative is to use attribute-based credentials as a basis for authentication and authorisation. Such credentials serve as a cryptographic container for attributes, that is, properties of the user, which are certified by an authority. With these attributes the user can be authenticated to access a resource or receive a service solely on the properties that are relevant for that specific resource or service.

In this thesis we discuss three attribute-based credential technologies for which we have developed efficient smart card implementations. These technologies are:

**Self-blindable Credentials** These credentials are based on elliptic curve cryptography with bilinear pairings. This technology shifts the computational burden to the terminal which makes a very compact smart card implementation possible. Unfortunately the support for elliptic curve cryptography on smart cards is limited to standard algorithms, which made it hard to develop other variants of this technology. This results in a minimal feature set compared to the other technologies.

**U-Prove** The U-Prove issuance and verification protocols are, respectively, based on Schnorr's blind signature scheme and zero-knowledge proofs. This technology offers the fastest implementation for attribute verification. With respect to privacy there is only one important drawback: U-Prove does not protect against linking multiple verification sessions to each other. This means that these credentials basically act as a pseudonym for the user.

**Identity Mixer** This technology is based on the Camenisch-Lysyanskaya signature scheme which provides a blind signature protocol, which can be used for credential issuance, and zero-knowledge proofs for attribute verification. The performance of this implementation is not the best among these technologies, but this technology provides a broad feature set and offers proper unlinkability. This makes it possible to use a credential multiple times without becoming traceable.

The goal of the research presented in this thesis has been to *develop efficient smart card implementations of attribute-based credentials* and *compare various cryptographic systems for attribute-based credentials.* This has resulted in a detailed

description and discussion of the technologies listed above and the smart card implementations[2] for each of these technologies, which are the most efficient implementations at the time of writing.

Furthermore, the successful development of these implementations laid the foundation for the IRMA project. This is an on-going research and development project focusing on attribute-based credentials and their use in practice. For more information concerning the IRMA project, please visit `https://www.irmacard.org/`.

[2]The source code of these implementations is available at `http://github.com/pimvullers/`.

# Preface

Five years have passed since I started my PhD program in Nijmegen. On the one hand this is a journey that one has to perform alone, on the other hand it is a journey one cannot finish alone. Therefore I would like to say thank you to everybody who helped me during this period. I don't want to deprive anybody, but there are some that I want to thank explicitly.

First, my supervisor, Bart, who supervised my PhD thesis, but also my daily activities. You have supported me in my research and the writing of my articles, but you have also promoted my results and supported me in gaining experience in teaching. Despite your busy schedule, there has always been time to discuss my daily worries. Bart, thanks.

I did not write the articles, on which this PhD thesis is based, alone. Hence, I would like to thank my co-authors, Wojciech, Gergely, Bart, Jaap-Henk and Lejla sincerely for their help. I also want to thank Wojciech and Erik for teaching me their knowledge and experience in programming smart cards, which has been useful throughout my research.

Furthermore I would like to thank my fellow PhD students for the good time I have had. At the research school IPA it has always been fun, in particular with Jeroen and Carst, my fellow students from my student days in Eindhoven, and Gergely and Wouter, my colleagues from Nijmegen, and of course the organisers of IPA, Tim and Meivan. At the university I shared a spacious office with many other PhD students. The joint work, but definitely also the breaks and conversations made it a nice place to work. I remember, in particular, the small security research projects together with Joeri and Gerhard.

During my PhD program I have had the opportunity to do an internship at Microsoft. This allowed me to gain experience and get an impression of a research department of a large company. Christian, thanks for making all the necessary arrangements and the supervision during my stay at Microsoft.

My research laid the foundations for the IRMA project which focuses on the use of attribute-based credentials in practice. This project took up a lot of time and I am glad that I did not have to do this work alone. Antonio, Bart, Gergely, Jaap-Henk, Maarten, Martijn, Roland, Ronny and Wouter, thanks.

I would also like to thank the members of my thesis committee for the time they have spent judging my PhD thesis. In particular I want to thank Sjouke, since he has also been the supervisor of my MSc project in Luxembourg.

All the effort spent in these years has obviously also affected my personal life. During the defence of my doctoral thesis my brothers Bas and Koen will take the

role of paranymphs. Koen also saved me a lot of time by designing the cover of this thesis. My parents, Jan and Gerrie, have always supported me throughout my academic career, even when they could not understand what I was working on.

Finally I want to say thank you to Marlou, for whom it was not always easy during these years (in particular during the writing of this thesis), but who always supported me to finish this work. Thanks.

*Pim Vullers*
*Nooitgedacht, October 2014*

# Contents

# Chapter 1

# Introduction

The world is moving into a digital era. Computers are becoming more and more intertwined in our daily lives. Many people spend time on the Internet, not just for fun or gathering information, but also for social interaction, shopping or on-line banking. Not only do our activities take place in a digital world, existing systems are also moving to digital alternatives. Paper train tickets are being replaced by electronic public transport cards. Identity documents, such as passports and identity cards, are equipped with electronic chips to hold digital copies of the identity data printed on the document. Sometimes even additional information is stored in these chips, such as fingerprints or other biometric data.

Unfortunately, most digital systems use a simple approach to identify entities, including users; they just associate them with a unique identifier. While this is convenient for bookkeeping, it also has a big drawback with respect to privacy. Using these unique identifiers, it is easy to trace the user. This was already the case for tracking activities on the Internet, but now real world actions can easily be traced as well through the use of public transport cards or digital identity documents.

In a security context, these unique identifiers are used to identify entities during authentication and/or authorisation processes, but in many use cases identification is not necessary during such a transaction. For instance, when you want to buy liquor, a merchant only needs to verify that you are of a certain age. The same holds when boarding a train; the public transport system only needs to know whether or not you are allowed to do so, and there is no direct need for the system to know exactly who you are.

A more privacy-friendly approach is possible by using only specific properties of the user, or *attributes*, as an alternative to identities. Instead of providing lots of identity information to a service provider, the user can just provide the required attributes, such that the service can be accessed without the user revealing his identity. For example, when you want to buy a bottle of wine you just prove that you are over 18 (or 21) years of age, which is sufficient to authorise the transaction, without revealing who you are. To be more precise, it does not matter who you are, where you live, or even what your age actually is. You are allowed to buy the wine, as long as you satisfy the property that you have reached a certain age. This illustrates that it is often more important *what* you are than *who* you are.

## 1.1 Attributes and Credentials

Within this thesis, an attribute will be understood as a property or statement concerning a person. Examples of attributes are:

- I am a student (or senior citizen)
- I am over 12 (or 18, or 21, or 65)
- I have a second class train pass
- My gender is male / female
- My loyalty status for company ... is bronze / silver / gold / ...
- My name is ...
- My date of birth is ...
- My address is ...
- My social security number is ...
- I own the bank account with number ...

Note that some of these attributes, like your bank account or social security number, can be used to uniquely identify you, while other attributes are not uniquely identifying: they apply to other people as well.

Informally, a person's identity can be seen as the collection of all attributes that hold for this person. In practice, many transactions can be based on a minimal set of attributes, namely exactly those attributes which are relevant or required to carry out the transaction. For example one can think of the following scenarios:

- If you wish to get a cheaper meal, show the student attribute, and for cheaper public transport you show that you are a senior citizen.
- If you buy an item in an on-line shop you need to reveal your bank account and address attributes, and possibly also your loyalty status attribute.
- When you buy a certain type of game or video on-line, you need to prove that you possess the attribute over 12, or over 18.

Note that in the last scenario, instead of using an attribute with the over 12 or over 18 statement, this property can also be proved based on the date of birth attribute. This is then called a *derived attribute.*

There are several cryptographic systems for dealing with identities based on attributes. Typically these systems distinguish attributes and credentials. Informally, a credential is a *cryptographic container of attributes.* As a first approximation, one can think of a credential as depicted in Figure 1.1. Examples of credentials are [AJ13]:

- An **address** credential, containing for example the street, zipcode, city and country attributes.
- A (citizen) **identity** credential, containing for example the name, gender, **date of birth**, and social security number attributes.
- A **student** credential, containing for example the student number, field of study, enrolment year and university/college attributes.
- A **festival** credential, containing for example the festival name and date/time attributes.

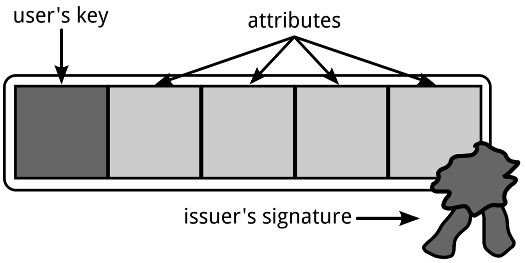


Figure 1.1: A visual representation of an attribute-based credential.

### 1.1.1 Credential Issuance and Verification

Credentials are *issued* and *verified*, whereas attributes can be *disclosed* or *proved* during verification. A credential is issued by an authority, the *issuer*, which can assess that the attribute statements in the credential hold for the individual they are issued to. This individual, the *user*, can subsequently use this credential to *prove* to another party, the *verifier*, that she has a certain qualification, competence, or property.

An issuer and a user together construct a new credential using the issuance procedure. First the user authenticates to the issuer in some reliable but (for this description) unspecified manner (which may be face-to-face). Once the authentication succeeds, the issuer collects the attributes for this user from its trusted sources. The user and issuer then carry out a cryptographic protocol in which the attributes are combined into a credential signed by the issuer. The resulting credential contains the attributes concerning the user and also the user's personal key (as depicted in Figure 1.1).

The fact that the attributes hold for the owner of a credential is guaranteed both by the issuer's signature and by the embedded personal key of the owner. The secret key embedded in the credential plays an essential role in the verification procedure of the credential, since it is supposed to be only known by the credential owner. Therefore this secret key, if properly protected, ensures that a credential cannot be transferred from one user to another.

### 1.1.2 Selective Disclosure of Attributes

A user may have several credentials, each containing some collection of attributes. When requesting a service from a service provider, the user is required to authenticate using one (or more) of her credentials. In the verification process the user can choose to only provide certain credentials; also, given a specific credential, the user may choose to reveal only a selection of the attributes contained in the credential. By doing this, authentication becomes more privacy friendly. This verification process is called *selective disclosure*, involving a verification protocol in which only a subset of the credential attributes is revealed to the verifier while the other attributes are only proved to be present in the credential. This allows a user to reveal only the necessary attributes and prove that the credential belongs to her. The service provider can verify all information that has been sent, including the issuer's signature.

The roles of a service provider and an issuer can also be combined: after verifying one credential, a new one can be issued. For instance, after verifying an over 18 attribute from an identity credential, a liquor shop might choose to issue a loyalty credential.

In this thesis we stick to a simplistic approach to selective disclosure in which an attribute index set $\mathcal{A}_D \subseteq \mathcal{A}$ determines which of the attributes $\{a_i\}_{i \in \mathcal{A}}$ contained in the credential will be revealed to the verifier. Note that more advanced proofs about attributes can be generated [IBM12]. For example, the over 18 statement derived from date of birth attribute mentioned above, or that the current date is within the validity period specified in a train pass attribute [Rog11]. This is, however, out of scope for this thesis.

### 1.1.3 Security and Privacy

The cryptographic nature of the credential-as-container concept includes the following four security aspects.

- The issuer's digital signature ensures *authenticity*: the credential originates from the issuer, and this issuer asserts that the attributes hold for the user.
- This signature also guarantees *integrity*: the attributes contained in the credential have not been altered since they were issued.
- A credential is *non-transferable* as it is bound to the secret key of the person involved in the issuance protocol. This secret key should be well protected, for instance via storage in the secure memory of a smart card with a PIN.
- A credential *hides* its content, so it does not reveal the attributes it contains, unless these attributes are revealed during verification.

Furthermore, a credential can protect the privacy of its owner through the following two cryptographic properties.

- *Issuer unlinkability* ensures that any information that the issuer gathers during the issuing procedure cannot be used to link a verification of the credential to its issuance.
- *Multi-show unlinkability* guarantees that when a credential is verified multiple times, these sessions cannot be linked.

The privacy of users is protected by these unlinkability properties even if the credential issuer and all verifiers collude. These properties can be achieved in a variety of ways, as can be seen by the different attribute-based credential technologies that have been proposed.

## 1.2 Attribute-based Credential Technologies

A number of technologies have been developed based on ideas described above, but the main focus has been on the cryptography that enables such systems and less on (efficient) implementations and their use cases. The implementations which have been developed are mainly for ordinary computers, while our research focuses on implementing such technologies on smart cards. This approach offers various new usage scenarios, like privacy-friendly public transport cards and identity documents, but also faces difficulties due to the limited capabilities of smart card platforms and hardware (see Section 1.3).

The first attribute-based credentials have been described by Chaum [Cha85] in 1984. In this section we introduce a number of recent technologies which provide attribute-based credentials, which use various cryptographic methods to achieve the security and privacy properties mentioned above.

### 1.2.1 Randomisable Certificates

Credentials based on randomisable certificates, such as Verheul's self-blindable credentials [Ver01], employ special cryptographic techniques enabling the certificate structures to be randomised using blinding factors while preserving their verifiability. The benefit of this approach is that the use of such credentials is untraceable. To achieve this, the users can blind their credentials before they are verified, such that two occurrences of the same credential cannot be recognised.

These credentials, as proposed by Verheul [Ver01], are discussed in detail in Chapter 3 as well as our efficient smart card implementations [BHJ$^+$10, HJV10] using elliptic curve cryptography.

### 1.2.2 Single-show Credentials

Another approach is to use single-show credentials in combination with a blind signature protocol. Here the issuance involves creating a blind signature which conceals the resulting credential from the issuer. Therefore, the verification instances of this credential cannot be related to the issuing phase. These credentials are called single-show since they do not provide the multi-show unlinkability property[1] and can therefore be linked when they are used multiple times. Hence these credentials serve as a pseudonym.

Examples of this approach are the credentials proposed by Brands [Bra00], which are used in Microsoft's U-Prove technology [PZ13], and the light-weight credentials described by Baldimtsi and Lysyanskaya [BL12].

A previous attempt to implement such technology on a smart card by Tews and Jacobs [TJ09], based on Brands' description [Bra00], resulted in a highly involved application with running times in the order of 5–10 seconds which make it not really usable in practice. Our smart card implementation [MV11] of U-Prove not only has a much better performance but is also, except for some minimal limitations, compatible with the development kits provided by Microsoft. We discuss this implementation and the U-Prove technology [PZ13] in detail in Chapter 4.

### 1.2.3 Multi-show Credentials

The use of zero-knowledge proofs allows a user to prove ownership of a credential without revealing the credential itself. This achieves multi-show unlinkability, as

[1]Multi-show unlinkability for these schemes can be realised by issuing multiple credentials for the same set of attributes which can later be verified independently.

the verifier does not see the credential. Camenisch and Lysyanskaya [CL01, CL03] combine such proofs with randomisation of the issuer's signature to provide issuer unlinkability. Their credential scheme was used as the basis for IBM's Identity Mixer technology [IBM12] and the direct anonymous attestation scheme [BCC04a] which has been adopted in the Trusted Platform Module specification [Tru07] as the method for remote authentication of a hardware module.

In 2009 Bichsel et al. [BCGS09] implemented Identity Mixer on a Java Card whereas Sterckx et al. [SGPV09] did the same for direct anonymous attestation. They provide the first proper implementations of attribute-based credentials on smart cards. The major drawback of these implementations is the running time of several seconds which is still too much for being really practical.

Our efficient implementation [VA13] of the Identity Mixer technology is described in Chapter 5, together with a detailed description of the underlying technology.

### 1.2.4 Shared Keys

The schemes that we have described so far are (to the best of our knowledge) the only candidates that provide privacy by design and could be implemented on a smart card. However, we should also briefly mention the approach used by the German national identity card (nPA)[2], where a limited form of (anonymous) attribute use is achieved by altering the existing elliptic curve based electronic identity protocols by sharing private keys across large batches[3] of cards [BKMN10]. The protocol itself provides restricted access to the card by means of the so-called card verifiable certificate mechanism [Bun10] and allows for selective disclosure of attributes, depending on the rights specified in the certificate (for example, a liquor store is only authorised to check for the `over 18` attribute). Signed attributes are partly anonymous because of the sharing of the signing keys between batches of cards, such that a signature cannot be linked to a single card.

## 1.3 Smart Cards

During our research we try to assess how fast privacy-friendly protocols can be executed when run on a modern smart card. Hence implementing our prototypes requires an open smart card platform that also provides the necessary cryptographic hardware support, as previous research [TJ09] clearly shows that, in terms of performance, purely software-based prototypes[4] are not sufficient for realistic use. In practice that leaves us with two possible smart card platforms, Java Card and MULTOS, described below.

Regardless of the software platform operating the card, all smart cards provide the same external functionality. A smart card is an embedded device that communicates with the environment through Application Protocol Data Units (APDUs), byte arrays formatted according to the ISO7816-4 specification [ISO05]. Most notably, the APDUs constrain the communication payload to roughly 256 bytes in each direction for a single APDU exchange. The permanent storage of the card (EEPROM

[2]`http://www.personalausweisportal.de/`
[3]The size of these batches is in the order of a million cards per batch.
[4]These prototypes are implemented without the use of dedicated cryptographic routines.

memory) is considered highly secure, accessible only through the APDU commands offered by the application, which in turn are subject to any authentication and secure messaging requirements that the card application may impose.

### 1.3.1 Java Card

Java Card [Che00] is a now well-established smart card platform based on a tailored, cut-down version of the Java platform. One of the main features of Java Card is software interoperability. This allows a developer to write a smart card application, or applet, in Java which can be executed on the Java Card virtual machine. The Java Card API can then be used as an interface to the (cryptographic) hardware of the smart card, making the applet (almost) fully independent of the underlying hardware and operating system of the actual smart card.

#### Virtual Machine

The Java Card virtual machine specification [Sun06b] defines a restricted subset of the Java programming language, though it preserves many of the object-oriented features including inheritance, interfaces, and exceptions. The specification also defines a Java-compatible virtual machine for smart cards which consists of two parts; one part external to the card and the other running on the card itself. The on-card virtual machine interprets the bytecodes and manages the classes and objects. The other part is a converter tool, that loads, verifies, and further prepares the Java classes in a card applet for on-card execution.

#### Memory Management

On a Java Card device, memory is the most valuable resource[5]. In most Java Card implementations a garbage collector is not available. When an object is created, the object and its contents are preserved in non-volatile memory (EEPROM), making it available across sessions. Access to the volatile memory (RAM) is provided through the Java Card API, which defines methods that allow you to create transient data storage at run-time.

In a Java Card environment a few rules should be taken into account to prevent wasting memory at run-time. Arrays and primitive types should be allocated at object creation time, and object creation should be minimised in favour of object reuse. All arrays and objects that an applet needs during its lifetime should be created all in one go, when an applet is installed, such that no additional dynamic memory allocation is needed once the applet is up and running. To promote reuse, objects should remain in scope or referenced for the life of the applet, and their state reset as appropriate before reuse.

[5]A typical modern smart card only has 36 to 144 KB of EEPROM for storing data and 4 to 8KB of RAM (which is only partially available to an application developer) for session data and computations.

### Application Programming Interface

The Java Card API is carefully designed to support the smart card environment and has several built-in security features. For example, it provides predefined Java classes for hardware supported cryptographic key storage (with possible internal encryption). To account for different hardware profiles of a card, parts of the Java Card API implementation are made optional. For example, our development cards based on NXP SmartMX chips support both RSA and elliptic curve cryptography in hardware and expose this functionality through the API, while other cards may only support RSA, in which case all method calls related to elliptic curve cryptography result in a Java exception.

This brings us to the main shortcoming of the Java Card platform from our point of view. The Java Card API is predefined and aimed at high-level functionality. For example, for RSA based cryptography it is only possible to generate keys of predefined RSA lengths (such as 512 and 1024 bits) and perform RSA operations according to standard specifications, such as PKCS #1 [RSA12]. The underlying mathematical operations, such as modular exponentiation, are not available to a developer. Since all of the protocols that we are interested in require access to such cryptographic operations (in large modulo prime and/or elliptic curve domains), this is a practical show stopper. Similar problems have been reported by others [BCGS09, SGPV09] regarding the implementation of cryptographic protocols on a Java Card. Even more, an efficient implementation of the e-passport standard [Bun10] on a Java Card also requires cryptographic routines not anticipated by the standard Java Card API. In this case, due to high demand, Java Card producers decided to enrich the Java Card API with proprietary extensions to support e-passport standards [NXP09]. But this only solves the problem for one application type and, moreover, makes the platform non-interoperable.

## 1.3.2 MULTOS

The goal of the MULTOS platform is to provide a secure hardware-independent execution platform for smart cards. To this end, they developed a specification for the execution and memory models, explained in more detail below, that all MULTOS implementations must provide. Besides this mandatory part of the specification there are also a number of optional elements, mostly concerning cryptographic functionality that may or may not be available on a specific hardware platform. An overview of which functionality is provided by which card can be found in the MULTOS implementation reference [MIR12].

### Execution Model

Applications on a MULTOS card are executed in a virtual machine, called the application abstract machine. The functionality of this virtual machine is defined by the MULTOS specification to assure that applications are portable, that is, independent of the actual chip used[6]. The application abstract machine is a stack machine that interprets instructions from the MULTOS executable language (MEL).

[6]Application portability can be limited due to specific memory requirements or dependencies on optional parts of the MULTOS specification.

### Memory Model

The virtual machine provides each application with its own memory space. Within an application the code space, residing on non-volatile EEPROM storage, and data space, divided over EEPROM (for persistent storage) and volatile RAM, are handled independently of each other. The memory of an application is protected by a strong firewall. This means that applications cannot access each others memory. The data of an application is divided over three distinct memory areas, listed below.

**Static memory** is the non-volatile storage for an application. It is private to the application and cannot be accessed by the terminal or any other application. MULTOS offers mechanisms to avoid corruption of the static memory area such that this data remains consistent.

**Public memory** is the volatile input/output buffer for an application. Incoming command APDUs are held in public memory and outgoing response APDUs are placed here. This buffer is also used to pass information from one application to another when delegation is used. MULTOS guarantees that data in this memory remains private to the running application until it exits or delegates to another application. This means that it can be used as temporary work space.

**Dynamic memory** is the volatile storage for an application. It is used to store session data, if any. The size of the session data area is fixed when an application is loaded onto a card and it depends on the number of variables declared. Furthermore, the dynamic memory contains the stack, which is the application's work area. As mentioned before, the application abstract machine operates as a stack machine, which means that this memory area is used to perform many functions (and provide input for these functions). The size of the stack is limited by the size of the session data in the dynamic memory. Therefore, applications have to ensure that their dynamic memory use does not exceed the the limit of the used chip [MIR12].

### Application Programming Interface

Similar to the Java Card API routines, some of the instructions are specified to be optional, mostly ones responsible for cryptographic operations. A particular MULTOS card may or may not support the optional instructions. For our implementation we used development cards based on the SLE66 and SLE78 chips from Infineon. These particular cards [MIR12] support a wide range of modulo arithmetic operations, a range which is sufficient to fully support all of the required calculations. The more low-level and flexible MULTOS API, as opposed to the less flexible and more high level Java Card API, is the main reason to choose the MULTOS platform for the prototype implementations of the cryptographic protocols discussed in this thesis.

For our prototypes we used the MULTOS C interface and bits of MEL assembly. For simple smart card applications the C interface seems to provide an easier programming environment than Java, and allows for a more flexible (byte-level) memory management. Although C programming platforms are not type safe by definition (as opposed to Java), per application memory safety is guaranteed by the MULTOS platform, regardless of the high-level language used during development.

## 1.4 Contributions

The goal of the research presented in this thesis has been to

*develop efficient smart card implementations of attribute-based credentials*

and

*compare various cryptographic systems for attribute-based credentials.*

This has resulted in a detailed description and discussion of the technologies listed below, which can be found in Chapters 3, 4, 5 and 6. Additionally, a large and essential part of this thesis consists of the efficient smart card implementations for each of these technologies. The implementations themselves will not be described in detail in this thesis; we refer the interested reader to the source code repositories mentioned below. Instead, we concentrate on performance results, and on general issues or problems that came up during the implementation work.

The implementations developed during this research are available as open source software, under the GNU General Public License[7], version 3, unless stated otherwise in the `LICENSE` file included in the root of each repository. The source code of these implementations can be found in their respective repositories, as specified below, at `https://github.com/pimvullers/`.

### Self-blindable Credentials

The third chapter is based on two papers, *Developing Efficient Blinded Attribute Certificates on Smart Cards via Pairings* [BHJ$^+$10] which is joint work with Lejla Batina, Jaap-Henk Hoepman, Bart Jacobs and Wojciech Mostowski, and *Privacy and Security Issues in e-Ticketing – Optimisation of Smart Card-based Attribute-proving* [HJV10] which is joint work with Jaap-Henk Hoepman and Bart Jacobs. I presented this work at the 9th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Applications (CARDIS 2010).

**Contribution** My contribution in this chapter is the development and analysis of the *first* efficient smart card implementation of the self-blindable credentials technology as well as the implementation of the corresponding host software. This consists of:

- the Java Card applet (`sbcred_javacard` repository[8]), which provides the cryptographic operations on the smart card;
- the terminal software (`sbcred_terminal` repository[9]), which provides the cryptographic operations for the issuer and verifier as well as the protocol interaction with the smart card; and
- an extension, (`bouncycastle-ext` repository[10]), to the Bouncy Castle cryptographic library[11], which adds support for elliptic curve bilinear pairings.

[7]`https://www.gnu.org/copyleft/gpl.html`
[8]`https://github.com/pimvullers/sbcred_javacard/`
[9]`https://github.com/pimvullers/sbcred_terminal/`
[10]`https://github.com/pimvullers/bouncycastle-ext/`
[11]`http://bouncycastle.org/java.html`

### U-Prove

The fourth chapter is based on the paper *Efficient U-Prove Implementation for Anonymous Credentials on Smart Cards* [MV11] which is joint work with Wojciech Mostowski. I presented this work at the 7th International ICST Conference on Security and Privacy in Communication Networks (SecureComm 2011).

**Contribution** My contribution in this chapter is the analysis of the efficient smart card implementation of the U-Prove technology as well as the development of the terminal software (`uprove_terminal` repository[12]), which builds upon the U-Prove SDK[13] and takes care of the protocol interaction with the smart card. Furthermore I provided the pseudo code that allowed my co-author to develop the MULTOS application (`uprove_multos` repository[14]), which provides the cryptographic operations on the smart card.

### Identity Mixer

The fifth chapter is based on the paper *Efficient Selective Disclosure on Smart Cards using Idemix* [VA13] which is joint work with Gergely Alpár. I presented this work at the 3rd IFIP WG 11.6 Working Conference on Policies and Research in Identity Management (IDMAN 2013).

**Contribution** My contribution in this chapter is the development and analysis of the efficient smart card implementation of the Identity Mixer technology, which is the first smart card implementation to include *selective disclosure* of the attributes, as well as the implementation of the corresponding host software. This consists of:

- the MULTOS application[15] (`idemix_multos` repository[16]), which provides the cryptographic operations on the card; and
- the terminal software (`idemix_terminal` repository[17]), which provides the protocol interaction between the smart card and our patched version of the Identity Mixer cryptographic library[18] (`idemix_library` repository[19]).

This implementation is the most complicated and challenging one in this thesis. Because of the complicated nature of the Identity Mixer operations it was, at first, not expected that a fast implementation could be realised on a smart card. The successful development of this implementation laid the foundation for the IRMA project. This is an on-going research and development project focusing on attribute-based credentials and their use in practice.

[12]`https://github.com/pimvullers/uprove_terminal/`

[13]`http://archive.msdn.microsoft.com/uprovesdkjava/`

[14]`https://github.com/pimvullers/uprove_multos/`

[15]This has evolved into the IRMA application (`https://github.com/pimvullers/irma_card/`) which provides additional functionality, like secure messaging and an interface which allows the user to manage the credentials stored on the card.

[16]`https://github.com/pimvullers/idemix_multos/`

[17]`https://github.com/pimvullers/idemix_terminal/`

[18]`https://prime.inf.tu-dresden.de/idemix/`

[19]`https://github.com/pimvullers/idemix_library/`

This thesis does not described the IRMA project in great detail but concentrates on the smart card implementations of the underlying attribute-based credential technologies. For more information concerning the IRMA project, please visit `https://www.irmacard.org/`.

# Chapter 2

# Cryptographic Preliminaries

This chapter provides some cryptographic background information which is relevant for the attribute-based credential systems which we describe in the next chapters. In particular we focus on public-key cryptography in which a key consists of a public and a private part. These key pairs are constructed such that deriving the private part of the key from the public part is equivalent to solving a computational problem that is considered extremely difficult.

## 2.1 RSA Cryptography

In an RSA-based cryptosystem [RSA78] a key pair consists of a public part $(n, e)$ and a private part $d$. The RSA modulus $n = p \cdot q$ is the product of two primes $p$ and $q$ and the public exponent $e$ is a value that satisfies $1 < e < \phi(n)$ and $gcd(e, \phi(n)) = 1$ where $\phi(n) = (p - 1)(q - 1)$ is Euler's totient function[1]. The private exponent $d$ is a value that satisfies $1 < d < \phi(n)$ and $e \cdot d = 1 \mod \phi(n)$, hence it can be computed as $d = e^{-1} \mod \phi(n)$. When $p$ and $q$ are know, this computation is easy, but computing $d$ based on just $n$ and $e$ is proved to be computationally equivalent to determining the prime factors $p$ and $q$ of $n$, which is known as the integer factorisation problem, a number-theoretic problem which is considered intractable for large integers.

### 2.1.1 Encryption Scheme

Such an RSA key pair can then be used to encrypt a message $m$ into an RSA ciphertext $c = m^e \mod n$ using Algorithm 2.1. Decryption of such a ciphertext using Algorithm 2.2 is based on the fact that

$$c^d = (m^e)^d = m^{e \cdot d} = m \mod n.$$

The problem of recovering a random message $m$ based on the ciphertext $m^e \mod n$ and public key $(n, e)$ is known as the *RSA problem*. This is equivalent

[1]Euler's totient function $\phi(n)$ computes the number of positive integers less then or equal to $n$ that are relatively prime to $n$. An integer $k$ is relatively prime to $n$ if $gcd(k, n) = 1$.

**Algorithm 2.1** Basic RSA encryption.

```
1: function RSA-ENCRYPT($(n, e), m$)
2:     $c \leftarrow m^e \mod n$
3:     return $c$
```

**Algorithm 2.2** Basic RSA decryption.

```
1: function RSA-DECRYPT($(n, e), c, d$)
2:     $m \leftarrow c^d \mod n$
3:     return $m$
```

to finding $e$th roots modulo $n$ which is assumed to be as difficult as the integer factorisation problem. Hence, the *RSA assumption* states that the probability that an attacker can solve the RSA problem is negligible.

Note that the algorithms described in this thesis are the basic textbook versions, which are vulnerable to a range of attacks [Hås86, Cop97]. To prevent such attacks, practical RSA implementations typically include some form of randomised padding into the value $m$ before encrypting it. For example, for encryption one would normally use the OAEP padding scheme from the PKCS #1 standard [RSA12]. The same holds for RSA signatures, as described below, where usually the PSS padding scheme [RSA12] is used to securely pad messages before signature generation.

### 2.1.2 Signature Schemes

In a similar fashion, this construction can also be used to create digital signatures. To generate an RSA signature with Algorithm 2.3, the signer computes the message digest $h = \textsc{Hash}(m)$ of the message to be signed $m$ using a cryptographic hash function Hash. This $h$, which serves as a fingerprint of the original message, is then raised to the private exponent. The result of this operation is the signature $s = h^d \mod n$ over the message $m$. Such an RSA signature can be verified using Algorithm 2.4. The verifier recovers the fingerprint $\hat{h} = s^e \mod n$ from the signature value $s$ using the public exponent $e$ and checks whether this matches with the message digest of the message $m$. If they match, the signature is valid, otherwise the signature is invalid. The security of this RSA signature scheme is also based on the RSA assumption.

**Algorithm 2.3** Basic RSA signature generation.

```
1: function RSA-SIGN($(n, e), m, d$)
2:     $h \leftarrow \textsc{Hash}(m)$
3:     $s \leftarrow h^d \mod n$
4:     return $s$
```

**Algorithm 2.4** Basic RSA signature verification.

```
1: function RSA-VERIFY($(n, e), m, s$)
2:     $\hat{h} \leftarrow s^e \mod n$
3:     if $\hat{h} \neq$ HASH($m$) then
4:         return INVALID
5:     return VALID
```

Some other signature schemes based on the RSA cryptosystem, such as the Camenisch-Lysyanskaya scheme [CL03] which is described in Section 5.1, only use the RSA modulus $n$ as the public part of the key, whereas the private part consists of the primes $p$ and $q$. This allows them to generate a fresh exponent $e$ for each signature which will then become part of the signature. Since an attacker can now control both the signature and the exponent, solving the RSA problem has become easier. Hence a stronger assumption is needed. This *strong RSA assumption* states that the probability that an attacker can solve the RSA problem is negligible, even when the attacker can chose the public exponent $e$.

## 2.2 Discrete Logarithm Cryptography

In a discrete logarithm-based cryptosystem [DH76, EG85] a key pair is accompanied with a description of the prime-order group in which the computations take place. As an example we use $(p, q, g)$, where $p$ is a prime, $q$ is a prime divisor of $p - 1$, and $g$ is a generator, with order $q$, of a subgroup of $\mathbb{Z}_p^*$. A private part of the key in such system is a random value $x$ and the corresponding public part is $h = g^x \mod p$. The problem of computing the private part $x = \log_g h$, based on the description of the group $(p, q, g)$ and the public part $h$, is known as the *discrete logarithm problem*.

### 2.2.1 Key Agreement Scheme

The first discrete logarithm-based scheme was the key agreement scheme by Diffie and Hellman [DH76]. In this scheme a key pair as described above can be used to compute a shared key with another party that uses the same group parameters. To this end, both parties generate such a key pair and send their public key to each other. Next, they compute the modular exponentiation of the received value and their private key, as depicted in Figure 2.1. The result of this computation can then be used as a shared key since

$$h_2^{x_1} = (g^{x_2})^{x_1} = g^{x_2 \cdot x_1} = k = g^{x_1 \cdot x_2} = (g^{x_1})^{x_2} = h_1^{x_2} \mod p$$

The problem of computing $k$, based on both public keys $h_1$ and $h_2$ and the group description $(p, q, g)$, is assumed to be as difficult as the discrete logarithm problem and is called the *(computational) Diffie-Hellman problem*. A related problem is to determine whether a value $y$ is created using $h_1$ and $h_2$, hence whether $y = g^{x_1 \cdot x_2}$, or not. This is known as the *decisional Diffie-Hellman problem*.

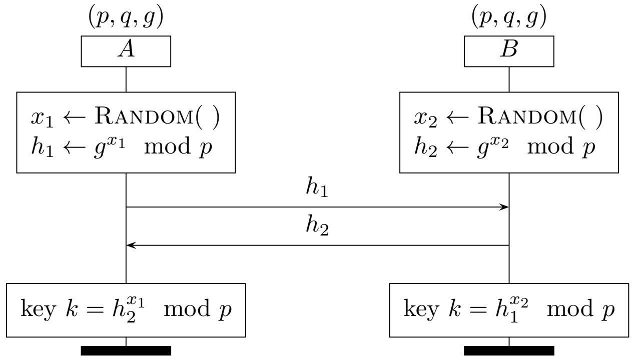

$(p, q, g)$
$A$
$x_1 \leftarrow \textsc{Random}(\ )$
$h_1 \leftarrow g^{x_1} \mod p$
$(p, q, g)$
$B$
$x_2 \leftarrow \textsc{Random}(\ )$
$h_2 \leftarrow g^{x_2} \mod p$
$h_1$
$h_2$
key $k = h_2^{x_1} \mod p$
key $k = h_1^{x_2} \mod p$

Figure 2.1: Diffie-Hellman key agreement protocol.

### 2.2.2 Encryption

The first encryption scheme based on discrete logarithms was proposed by ElGamal [EG85]. In order to encrypt a message $m$, the user generates a random value $r$ and commits to it by computing $c_1 = g^r \mod p$. The value $r$ is then used to randomise the public key which is multiplied with the message to obtain the encrypted message $c_2 = m \cdot h^r \mod p$. The resulting ciphertext consists of both $c_1$ and $c_2$ (see Algorithm 2.5). In contrast to the RSA encryption scheme, the ElGamal encryption algorithm produces a different ciphertext each time although the inputs remain the same.

The message $m$ can be recovered from a ciphertext $(c_1, c_2)$ by dividing $c_2$ by $c_1^x = g^{r \cdot x} = h^r \mod p$, as described in Algorithm 2.6. An attacker, who does not have the private key $x$, must determine $h^r = g^{x \cdot r} \mod p$ based on the public key $h = g^x \mod p$ and the commitment $c_1 = g^r \mod p$, that is, he must solve the Diffie-Hellman problem.

**Algorithm 2.5** ElGamal encryption.

```
1: function ElGamal-encrypt($(p, q, g), h, m$)
2:     $r \leftarrow$ Random
3:     $c_1 \leftarrow g^r \mod p$
4:     $c_2 \leftarrow m \cdot h^r \mod p$
5:     return $(c_1, c_2)$
```

**Algorithm 2.6** ElGamal decryption.

```
1: function ElGamal-decrypt($(p, q, g), (c_1, c_2), x$)
2:     $m \leftarrow c_2 \cdot c_1^{-x} \mod p$
3:     return $m$
```

### 2.2.3 Signatures

Many signature schemes have been based on the discrete logarithm problem, for example the Chaum-Pedersen scheme which is described in Section 3.1.1, but also the ElGamal signature scheme [EG85] of which a variant, known as the Digital Signature Algorithm, is part of the Digital Signature Standard of the United States government. Like ElGamal's encryption scheme, the signature $(s_1, s_2)$ consists of two parts generated using Algorithm 2.7. The first is a commitment to the randomisation value $r$, whereas the second part commits to the fingerprint $c$ of the message $m$ that is to be signed. In the unlikely case that the resulting $s_2$ is 0, these steps have to be repeated to obtain a usable signature.

This signature can be verified using Algorithm 2.8. We cannot recover the message fingerprint from the signature, as was the case with RSA, but we can check it according to the following equation:

$$h^{s_1} \cdot s_1^{s_2} = g^{x \cdot s_1} \cdot g^{r \cdot s_2} = g^{x \cdot s_1} \cdot g^{r \cdot r^{-1} \cdot (c - x \cdot s_1)} = g^{x \cdot s_1} \cdot g^{c - x \cdot s_1} = g^c \mod p$$

**Algorithm 2.7** ElGamal signature generation.

```
1: function ElGamal-sign($(p, q, g), m, x$)
2:     $c \leftarrow$ Hash($m$)
3:     $r \leftarrow$ Random( )
4:     $s_1 \leftarrow g^r \mod p$
5:     $s_2 \leftarrow r^{-1} \cdot (c - x \cdot s_1) \mod (p-1)$
6:     return $(s_1, s_2)$
```

**Algorithm 2.8** ElGamal signature verification.

```
1: function ElGamal-verify($(p, q, g), m, (s_1, s_2), h$)
2:     $c \leftarrow$ Hash($m$)
3:     if $h^{s_1} \cdot s_1^{s_2} \neq g^c \mod p$ then
4:         return Invalid
5:     return Valid
```

## 2.3 Elliptic Curve Cryptography

In the previous section we described the discrete logarithm cryptography in a setting of a prime order subgroup of $\mathbb{Z}_p^*$. These techniques can, however, be applied to any finite cyclic group. Another popular setting for implementing discrete logarithm systems is the elliptic curve setting which we describe below.

Note that it is common to use additive notation in an elliptic curve setting, whereas multiplicative notation is typically used to denote operations in prime order subgroups.

### 2.3.1 Elliptic Curves

A finite field containing $q$ elements, where $q = p^k$ is a prime power, is denoted as $\mathbb{F}_q$. An example of such a field is $\mathbb{Z}_p$, but more complex finite fields can also be used for elliptic curves. An elliptic curve $E$ over $\mathbb{F}_q$ is defined by the equation

$$E : y^2 = x^3 + a \cdot x + b \tag{2.1}$$

where $a, b \in \mathbb{F}_q$ are the curve parameters that satisfy $4 \cdot a^3 + 27 \cdot b^2 \neq 0$. This condition is required for E to be non-singular, as required for cryptographic applications. The set of all points $P = (x, y)$ on this curve $E$ is denoted as

$$E(\mathbb{F}_q) = \{(x, y) \in \mathbb{F}_q \times \mathbb{F}_q : y^2 = x^3 + a \cdot x + b\} \cup \{\infty\}$$

where $\infty$ is a special point called the *point at infinity*.

On these points three basic operations can be defined: negation, addition and doubling. These operations are detailed below and visualised geometrically in Figure 2.2.

1. *Point negation*, the *negative* $R = -P$, of the point $P$ is computed according to Algorithm 2.9. Geometrically, this is the reflection of point $P$ in the $x$-axis.

**Algorithm 2.9** Elliptic curve point negation: $R = -P$

```
1: function ECPointNegation($P$)
2:     if $P = \infty$ then
3:         return $\infty$
4:     $(x_P, y_P) \leftarrow P$
5:     return $(x_P, -y_P)$
```

2. *Point addition*, the *sum* $R = P + Q$, of the points $P$ and $Q$ is computed using Algorithm 2.10. In geometry, this is the reflection in the $x$-axis of the intersection of the curve and the line through the points $P$ and $Q$.

**Algorithm 2.10** Elliptic curve point addition: $R = P + Q$

```
1: function ECPointAddition($P$, $Q$)
2:     if $P = \infty$ then
3:         return $Q$
4:     if $P = Q$ then
5:         return $2 \cdot P$
6:     if $P = -Q$ or $Q = \infty$ then
7:         return $\infty$
8:     $(x_P, y_P) \leftarrow P$
9:     $(x_Q, y_Q) \leftarrow Q$
10:    $x \leftarrow \left(\frac{y_Q - y_P}{x_Q - x_P}\right)^2 - x_P - x_Q \mod q$
11:    $y \leftarrow \left(\frac{y_Q - y_P}{x_Q - x_P}\right)(x_P - x_R) - y_P \mod q$
12:    return $(x, y)$
```

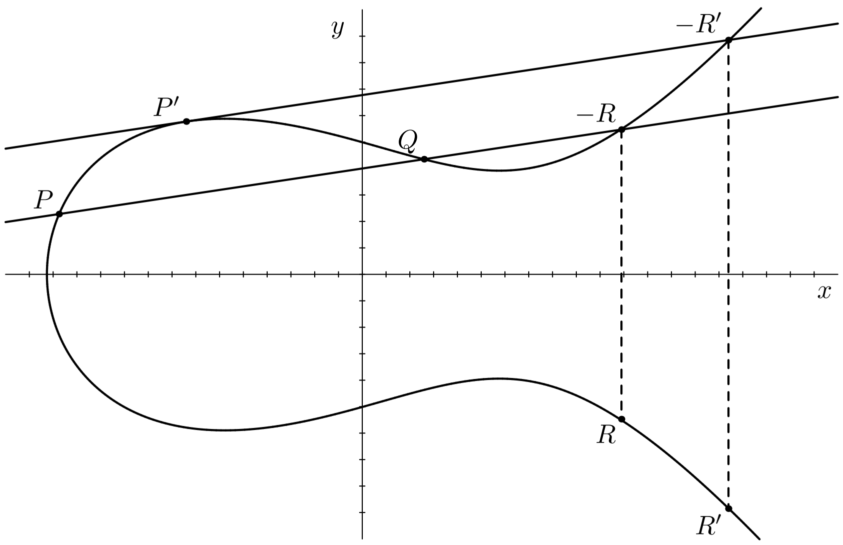


Figure 2.2: Geometric visualisation of point addition ($R = P + Q$), point doubling ($R' = 2P'$) and point negation ($-R$) on an elliptic curve.

3. *Point doubling*, the *double* $R = 2 \cdot P$ is computed using Algorithm 2.11. Geometrically, this is the reflection in the $x$-axis of the intersection of the curve and the tangent line to the curve at the point $P$, see Figure 2.2.

**Algorithm 2.11** Elliptic curve point doubling: $R = 2 \cdot P$

```
1: function ECPointDoubling($P$)
2:     if $P = \infty$ then
3:         return $P$
4:     $(x_P, y_P) \leftarrow P$
5:     $x_R \leftarrow \left(\frac{3x_P^3 + a}{2y_P}\right)^2 - 2x_P \mod q$
6:     $y_R \leftarrow \left(\frac{3x_P^3 + a}{2y_P}\right)(x_P - x_R) - y_P \mod q$
7:     return $(x_R, y_R)$
```

Furthermore, based on these basic operations we can define *point multiplication* as the multiplication of a point $P$ with an integer $k$, which is denoted as $k \cdot P$. To compute this multiplication various methods exist. A basic solution is the repeated-double-and-add method given in Algorithm 2.12. The inverse of this operation is to find an integer $k$ such that $Q = k \cdot P$ for given points $P$ and $Q$. This is a hard problem which is know as the *elliptic curve discrete logarithm problem*.

Finally, if we know the $x$-coordinate of a point on the curve, the square of the corresponding $y$-coordinate is known, namely as defined in (2.1). By taking the square root of $x^3+a\cdot x+b$ we find either $y$ or $-y$. This method of *point reconstruction* forms the basis of *point compression*, for compact representation of points.

**Algorithm 2.12** Elliptic curve point multiplication: $R = k \cdot P$

```
1: function ECPointMultiplication($k$, $P$)
2:     $(k_{l-1}, \ldots, k_1, k_0) \leftarrow k$          ▷ binary representation of $k$
3:     $R \leftarrow \infty$
4:     for $i$ from 0 to $l - 1$ do
5:         if $k_i = 1$ then
6:             $R \leftarrow R + P$
7:         $P \leftarrow 2P$
8:     return $R$
```

The points on the elliptic curve $E$ form a finite cyclic group with point addition as the group operation and the point at infinity as the zero-element. Hence, such a group can be used to implement discrete logarithm-based cryptography. An example to show how easy it is to translate the algorithms and protocols for discrete logarithm-based cryptography to the elliptic curve setting is given in Figure 2.3. Like its counterpart from Figure 2.1, this version of the Diffie-Hellman key agreement protocol also computes a shared key $K = x_1 \cdot x_2 \cdot P$. The only difference is that $K$ is now a point instead of an integer.

## 2.3.2 Pairings

Besides an alternative implementation for discrete logarithm-based cryptography, elliptic curves also offer additional functionality, such as efficiently computable bilinear pairings.

A bilinear pairing is a map of the form $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ where $\mathbb{G}_1$ and $\mathbb{G}_2$ are typically additive groups and $\mathbb{G}_T$ is a multiplicative group. Bilinearity means that the map is linear in both components. This bilinearity property can be written as follows:

$$\begin{array}{rcl} e(P + P',\ Q) & = & e(P,\ Q) \cdot e(P',\ Q) \\ & \text{and} & \\ e(P,\ Q + Q') & = & e(P,\ Q) \cdot e(P,\ Q') \end{array}$$

As a result, $e(n \cdot P,\ m \cdot Q) = e(P,\ Q)^{n \cdot m}$.

Pairings are used for many (new) cryptographic protocols [BSS05], such as short signatures [BLS04], three-party one-round key agreement [Jou04], identity based encryption [BF01] and anonymous credentials [CL04]. There are different pairings that can be used for this kind of cryptography, for example the Weil pairing, Tate pairing, ate pairing and the recent R-ate pairing [Ver09]. For all these pairings one often uses specific cyclic subgroups of a curve $E(\mathbb{F}_{p^k})$ as $\mathbb{G}_1$ and $\mathbb{G}_2$ and $\mathbb{F}^*_{p^k}$ as $\mathbb{G}_T$.

### Barreto-Naehrig Curves

Pairing-friendly elliptic curves are curves with a small embedding degree and a large prime-order subgroup [FST10]. In 2005, Barreto and Naehrig discovered a new method for constructing pairing friendly elliptic curves of prime order over a prime field [BN06]. More precisely, Barreto-Naehrig curves are defined over $\mathbb{F}_p$ where $p = p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$ for $u \in \mathbb{Z}$ such that $p$ is prime. The order

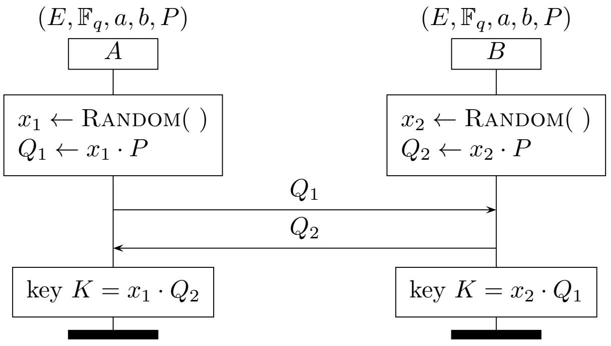

$(E, \mathbb{F}_q, a, b, P)$
$A$
$(E, \mathbb{F}_q, a, b, P)$
$B$
$x_1 \leftarrow \textsc{Random}(\ )$
$Q_1 \leftarrow x_1 \cdot P$
$x_2 \leftarrow \textsc{Random}(\ )$
$Q_2 \leftarrow x_2 \cdot P$
$Q_1$
$Q_2$
key $K = x_1 \cdot Q_2$
key $K = x_2 \cdot Q_1$

Figure 2.3: Elliptic curve version of the Diffie-Hellman key agreement protocol.

of such curve is a prime $n$ where $n = n(u) = 36u^4 + 36u^3 + 18u^2 + 6u + 1$. Hence, a Barreto-Naehrig curve is constructed by generating integers $u$ until both $p(u)$ and $n(u)$ are prime numbers.

## 2.4 Proofs of Knowledge

A cryptographic concept that is frequently used in attribute-based credential is a *proof of knowledge*. The goal of such proof is for the user, or *prover*, to convince the verifier of a given statement. For example, the often used challenge-response construction in which a user has to sign or decrypt a challenge to authenticate herself, is a proof that she knows the private key corresponding to the public key used by the verifier.

To describe such proofs of knowledge we use the notation introduced by Camenisch and Stadler [CS97]. For example,

$$PK\{(\alpha) : h = g^\alpha \mod p\}$$

denotes a proof of knowledge of a value $\alpha$ such that $h = g^\alpha \mod p$, that is, a proof of knowledge of the private part of a discrete logarithm key pair.

### 2.4.1 Zero-knowledge Protocols

In this section we describe zero-knowledge protocols as a way to convince a verifier. The protocols have the property that no matter what a verifier does, he will not be able to extract any useful information from the user. More precisely, the term *zero-knowledge* refers to the fact that whatever information the verifier learns from the user, that information could have been generated by the verifier on its own, without the assistance of the user. However, a verifier that actually carried out the protocol will be convinced that the user has the specified knowledge, in our example, the private key.

A well-known example of a zero-knowledge protocol is Schnorr's identification protocol [Sch91], which proves knowledge of a discrete logarithm. This protocol is depicted in Figure 2.4. To prove that the user knows the private key, she first

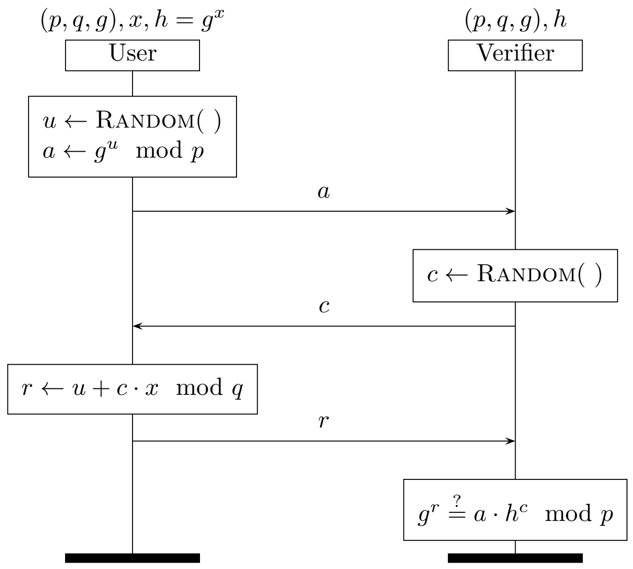

$(p, q, g), x, h = g^x$
User
$(p, q, g), h$
Verifier
$u \leftarrow \textsc{Random}(\ )$
$a \leftarrow g^u \mod p$
$a$
$c \leftarrow \textsc{Random}(\ )$
$c$
$r \leftarrow u + c \cdot x \mod q$
$r$
$g^r \overset{?}{=} a \cdot h^c \mod p$

Figure 2.4: Schnorr's zero-knowledge identification protocol.

commits to a random value $u$ and sends the commitment $a$ to the verifier. The verifier then generates a challenge $c$ at random and sends this to the user, which computes the response $r$ based on the challenge. Finally, the verifier checks whether $g^r = a \cdot h^c \mod p$.

The verifier does not learn anything from such a conversation $(a, c, r)$, since he could have computed such a triple himself by choosing $c$ and $r$ at random and computing $a = g^r \cdot h^{-c} \mod p$. This means that the zero-knowledge property holds for this protocol. Another important property is soundness, which guarantees that the user actually knows the the secret. Suppose that given a single commitment $a$ the user is able to respond to two different challenges, hence generating two conversations $(a, c, r)$ and $(a, c', r')$ where $c \neq c'$. Then, from $g^r = a \cdot h^c \mod p$ and $g'^r = a \cdot h^{c'} \mod p$ it follows that

$$a = g^r \cdot h^{-c} \mod p \quad \text{and} \quad g^{r'} = g^r \cdot h^{-c} \cdot h^{c'} \mod p$$

which implies that

$$g^{r'-r} = h^{c'-c} \mod p.$$

Hence, $h = g^{\frac{r'-r}{c'-c}} \mod p$ which means that the user actually knows the private key $x$, since

$$x = \frac{r'-r}{c'-c} \mod q.$$

A similar protocol [Cam07, Section 3.5], depicted in Figure 2.5, can also be constructed for groups in which the order of the (sub)group is not known to all parties. This is for instance the case in an RSA setting where the order of the group is only known by the party that knows the primes $p$ and $q$. As a result the user cannot perform the modular reduction using the order of $g$ when computing

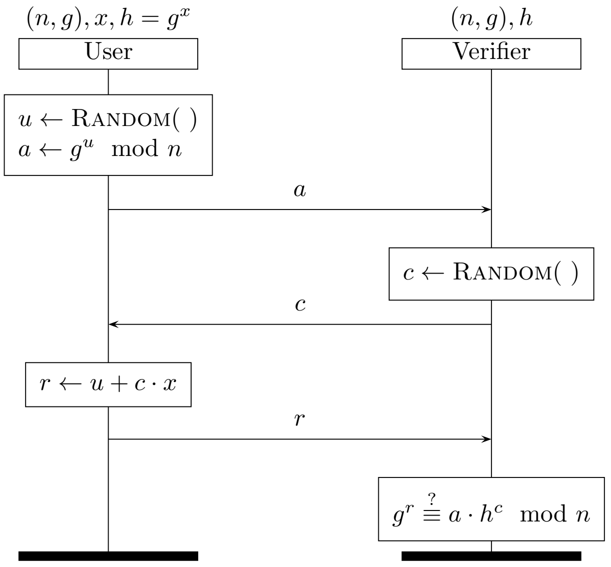


Figure 2.5: Schnorr's zero-knowledge protocol for groups of unknown order.

the response. This means that the response no longer hides the secret $x$ as it is not distributed uniformly. Therefore the user must choose a significantly larger[2] random value $u$ such that $a$ is distributed statistically close to uniform over the subgroup generated by $g$ and $x$ is statistically hidden in $r$. Hence, this protocol is called *statistical zero-knowledge* [Poi00].

Using these basic building blocks, more complex protocols can be constructed. For example, a proof for $PK\{(\dots) : P_1 \ \wedge \ P_2\}$, where the statements $P_1$ and $P_2$ do not share any of the variables of which the knowledge is being proved, can be built by constructing individual proofs for $P_1$ and $P_2$ using the same challenge $c$ for both protocols. Alternatively, when $P_1$ and $P_2$ do share a variable of which the knowledge is being proved, a proof can be built by constructing proofs for $P_1$ and $P_2$ individually, while using the same challenge $c$ for both protocols as well as the same random value $u$ and response $r$ for the shared variable,

### 2.4.2 Zero-knowledge Proofs

The Fiat-Shamir heuristic [FS87] can be used to transform a zero-knowledge protocol into a non-interactive zero-knowledge proof. This is often used to translate a zero-knowledge protocol into a signature scheme, or to reduce the communication overhead of the interactive protocols. To make a zero-knowledge protocol non-interactive the challenge $c$ is not retrieved from the verifier but computed as follows:

$$c \leftarrow \textsc{Hash}(m, a)$$

[2]For instance, the length of the random value $u$ should be 80-bits longer than the combined lengths of the modulus $n$ and the challenge $n$. This in contrast to just the length of the prime order $q$.

where HASH is a cryptographic hash function, $m$ is some message to be included (to be signed) and $a$ is the commitment. Both the commitment $a$ and the response $r$ are calculated as usual. The result is a signature $(c, r)$ on $m$. This can be verified by checking whether the following holds:

$$c = \textsc{Hash}(m, \hat{a})$$

where $\hat{a} = g^r \cdot h^{-c}$. If the proof $(c, r)$ is valid, this holds since:

$$\hat{a} = g^r \cdot h^{-c} = g^{u+c\cdot x} \cdot h^{-c} = g^{u+c\cdot x} \cdot (g^x)^{-c} = g^u = a \mod p.$$

Such a non-interactive zero-knowledge proof is often called a signature proof of knowledge, because of the message that is included in the proof. To describe such proof we use the following notation:

$$SPK\{(\alpha) : h = g^\alpha \mod p\}(m)$$

where $m$ is the message that is included in the non-interactive proof of knowledge of the private key corresponding to the public key $h$.

# Chapter 3

# Self-blindable Credentials

The drawback of using regular public-key certificates, like X.509 certificates [ISO08], is that they are inherently traceable. This is caused by the fixed public key and the signature that are contained in such a certificate. The public key can be used as a unique identifier for the user and is included in the signature, making that identifying as well.

To circumvent this problem Verheul [Ver01] proposes a variant [BLS01, BLS04] of the Chaum-Pedersen signature scheme [CP93] which allows these values to be randomised, or *blinded*, such that they are no longer traceable, while the signature can still be verified for its authenticity. This signature scheme can then be used to implement a credential system which allows the users themselves to blind their credentials in order to prevent traceability.

In this chapter we describe the underlying signature schemes and the resulting credential system as well as our smart card implementations [BHJ+10, HJV10].

## 3.1 Self-blindable Signatures

Chaum and Pedersen [CP93] describe a basic signature scheme which is intended to be used in combination with smart cards. This scheme originally operates in the discrete logarithm setting, but is translated to the elliptic curve setting by Boneh, Lynn and Shacham [BLS01, BLS04] and Verheul [Ver01]. The elliptic curve variant of this scheme led to the construction of self-blindable signatures.

### 3.1.1 Chaum-Pedersen Signature Scheme

The public key in this scheme is a value $h$ together with a description of the prime-order group in which the computations take place. As an example we use $(p, q, g)$, which denotes a subgroup of $\mathbb{Z}_p^*$ of prime-order $q$ with generator $g$. The corresponding private key is $x = \log_g h$.

The signature $z$ over a message $m$ is a single exponentiation with the private key together with a proof that $\log_g h = \log_m z$. This signature can be generated using Algorithm 3.1. Verification of the signature consists of checking the proof according to Algorithm 3.2. When the proof is correct, the verifier is assured that

**Algorithm 3.1** Generate a Chaum-Pedersen signature.

```
1: function CP-SIGN($m, (p, q, g), x$)
2:     $z \leftarrow m^x \mod p$
3:     $a \leftarrow g^s \mod p$
4:     $b \leftarrow m^s \mod p$
5:     $c \leftarrow$ HASH($m, z, a, b$)
6:     $r \leftarrow s + c \cdot x \mod q$
7:     return $(z, a, b, r)$
```

**Algorithm 3.2** Verify a Chaum-Pedersen signature.

```
1: function CP-VERIFY($m, (z, a, b, r), (p, q, g), h$)
2:     $c \leftarrow$ HASH($m, z, a, b$)
3:     if $g^r \neq a \cdot h^c \mod p$ then
4:         return INVALID
5:     if $m^r \neq b \cdot z^c \mod p$ then
6:         return INVALID
7:     return VALID
```

the message is signed using the private key corresponding to the public key used for the verification.

### 3.1.2 Boneh-Lynn-Shacham Signature Scheme

Since the signature scheme of Chaum and Pedersen operates in the discrete logarithm setting it can also be used with elliptic curve cryptography. Boneh, Lynn and Shacham [BLS01, BLS04] present a signature scheme that works on elliptic curves with bilinear pairings (see Section 2.3.2) and resembles the scheme by Chaum and Pedersen, but omits the equality proof in order to achieve short signatures.

At the same time Verheul [Ver01] gives a similar description of this proofless variant of the Chaum-Pedersen scheme for groups in which the decisional Diffie-Hellman problem is easy while the computational Diffie-Hellman and discrete logarithm problems are hard. Elliptic curves with bilinear pairings provide such groups. The proof of equality can in this case be substituted by a bilinear pairing equation to be checked by the verifier.

The public key is a point $Q = x \cdot P_2$ on the elliptic curve $E_2$ together with a description of that curve, of which $P_2$ is the generator. The private key in this scheme is the scalar value $x$. To sign a message $m$ the signer transforms it into a point $P_m$ on the curve $E_1$ which is simply multiplied with the private key as described in Algorithm 3.3.

To verify the signature the verifier has to perform the same check as with the Chaum-Pedersen scheme, that is, whether the private key used to generate the signature corresponds to the public key used for the verification. Instead of using a proof,

**Algorithm 3.3** Generate a Boneh-Lynn-Shacham signature.

```
1: function BLS-SIGN($m, E_1, x$)
2:     $P_m \leftarrow$ HASHTOPOINT($E_1, m$)
3:     $Z \leftarrow x \cdot P_m$
4:     return $Z$
```

**Algorithm 3.4** Verify a Boneh-Lynn-Shacham signature.

```
1: function BLS-VERIFY($m, Z, E_1, Q$)
2:     $P_m \leftarrow$ HASHTOPOINT($E_1, m$)
3:     if $e(P_m, Q) \neq e(Z, P_2)$ then
4:         return INVALID
5:     return VALID
```

this can be achieved by checking the following equation (as used in Algorithm 3.4):

$$e(P_m, Q) = e(P_m, x \cdot P_2) = e(x \cdot P_m, P_2) = e(Z, P_2) \tag{3.1}$$

where $e : E_1 \times E_2 \to G$ is a bilinear pairing as described in Section 2.3.2.

Verheul [Ver01] points out a powerful aspect about these signatures: they are invariant under blinding. When this signature is used to sign points on the curve, instead of arbitrary messages, there is no need to transform the message into a point, such that the HASHTOPOINT operation can be omitted. Now the user can choose a random number $b$ as blinding factor, and multiply both the message $P_m$ and signature $Z$ with this factor. The resulting pair $(b \cdot P_m, b \cdot Z)$ can still be verified using the verification equation (3.1):

$$e(b \cdot P_m, Q) = e(b \cdot P_m, x \cdot P_2) = e(b \cdot x \cdot P_m, P_2) = e(b \cdot Z, P_2)$$

Hence the signature remains valid. Since the user can perform this blinding all by itself, Verheul calls these signatures *self-blindable*.

## 3.2 Verheul's Self-blindable Certificates

Verheul proposes to use the self-blindable signatures to construct public-key certificates which allow the user to randomise its key pair and the corresponding certificate. Such certificates can be used to circumvent traceability based on the public key or the signature. A certificate of a user's public key $P_U$ from an identity provider with public verification key $P_{ID}$ takes the form

$$\{P_U, Sig(P_U, s_{ID})\}, \tag{3.2}$$

where $s_{ID}$ is the private signing key of the identity provider corresponding to $P_{ID}$.

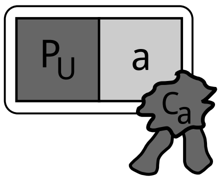


Figure 3.1: A visual representation of a self-blindable credential.

### 3.2.1 Attribute Certificates

Attribute certificates are digital certificates that bind attributes to users known by a public key. Proof of possessing an attribute is given by proving possession of the private key related to the public key referenced in the certificate.

$$\{P_U, [Sig(P_U, s_a), Cert(Q_a, \text{"Attribute statement"})]\}$$

Here, the public key $P_U$ of the user is signed using a private key $s_a$ of the issuer of the attribute. The corresponding public key $Q_a$ of the issuer is included in a (conventional PKI) certificate which contains the attribute statement corresponding to this public key. In the context of attribute-based credentials such a self-blindable attribute certificate can be considered as a self-blindable credential.

When we consider only a single attribute statement per public key, the certificates with the statements can be stored in a public database. This allows for more compact credentials, since this information can be omitted from the credential, given that the corresponding public key can be identified through some reference. Such compact credentials are beneficial for devices with limited storage capacities, like smart cards. To summarise, a self-blindable credential, as depicted in Figure 3.1, will consist of:

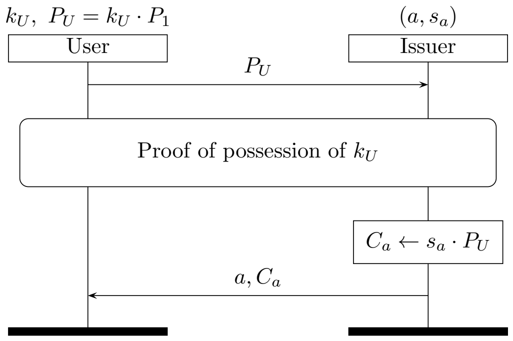
- the user's public key $P_U$,
- a reference to the attribute statement $a$, and
- a signature $C_a = Sig(P_U, s_a) = s_a \cdot P_U$ over the public key using the attribute signing key $s_a$.

The database will then contain:

- a reference $a$,
- the public verification key $Q_a$ corresponding to $s_a$, and
- the actual attribute statement.

### 3.2.2 Alternative Public Key Construction

Verheul also describes an alternative construction [Ver01] which provides more robustness. This construction is based on Okamoto's public key scheme [Oka93] where the key is constructed using two generators and two private key values instead of a single generator and private key value.



$k_U,\ P_U = k_U \cdot P_1$
$(a, s_a)$
User
Issuer
$P_U$
Proof of possession of $k_U$
$C_a \leftarrow s_a \cdot P_U$
$a, C_a$

Figure 3.2: Issuance of self-blindable credentials.

The self-blindable certificates still have the same structure as given in (3.2), except that the public key of the user is constructed slightly different. While this scheme is very similar, we could not implement this variant due to the restrictions of the Java Card API which we discuss in Section 3.5. Hence, in the remainder of this work we only consider the simple construction described above.

## 3.3 Credential Issuance

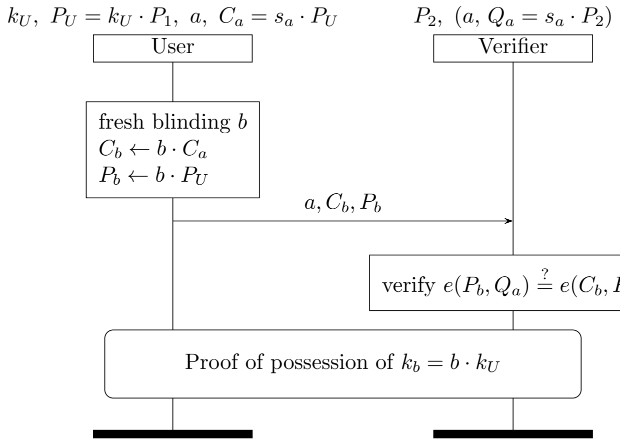
To obtain a self-blindable credential from a credential issuer the user must provide the issuer with its public key and prove possession of the corresponding private key. This public key can be signed by an identity provider in order to increase the trust level of the system. When the credential issuer has verified the authenticity of the user he will verify eligibility for the requested attribute. Once this has been verified the issuer signs the user's public key using the private key corresponding to the requested attribute and send this signature to the user as depicted in Figure 3.2.

## 3.4 Credential Verification

When the user wants to utilise an attribute stored in a credential it has to show the credential to a verifier. To prevent the verifier from tracing her based on the public key the user first blinds her key pair and the credential as shown in Figure 3.3. Next, she sends the results from this blinding operation to the verifier and proves possession of the (blinded) private key. This last step is easily achieved by signing a challenge received from the verifier using this private key. The verifier can then verify this signature using the public key it received earlier.

## 3.5 Smart Card Implementation

To understand the practical limitations and estimate the performance of our protocols, we implemented the protocol from Figure 3.3 in Java (for the terminal-side application) and Java Card [Che00] (for the card-side application). Our implementation involves the following components:



$k_U,\ P_U = k_U \cdot P_1,\ a,\ C_a = s_a \cdot P_U$
$P_2,\ (a, Q_a = s_a \cdot P_2)$
User
Verifier
fresh blinding $b$
$C_b \leftarrow b \cdot C_a$
$P_b \leftarrow b \cdot P_U$
$a, C_b, P_b$
verify $e(P_b, Q_a) \stackrel{?}{=} e(C_b, P_2)$
Proof of possession of $k_b = b \cdot k_U$

Figure 3.3: Verification of self-blindable credentials.

**Java Card applet**[1,2]

The protocol on the card-side is implemented as a Java Card applet and loaded onto a development smart card. This implementation will be explained in more detail below.

**Bouncy Castle cryptographic library**[3] **with an extension**[4] **for pairings**

The Bouncy Castle library is a collection of cryptographic APIs for the C# and Java programming languages. It provides full support for elliptic curve cryptography and an interface to the common Java Cryptography Extension API. However, it does not implement bilinear pairings or elliptic curves over fields other than $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$. Thus we have added our own implementations of $\mathbb{F}_{p^2}$ and $\mathbb{F}_{p^{12}}$, and the Tate, ate, and R-ate pairings.

This work greatly benefited from an ate pairing in Java that Paulo Barreto kindly made available[5], and algorithms published by Hankerson et al. [HMS09]. To minimise maintenance overhead we strived to keep our extensions purely *on top* of the Bouncy Castle library, that is, we did not change anything in the original library. This allows for easy integration in newer library releases.

**Terminal application**[6]

The protocol on the terminal-side is implemented using the aforementioned cryptographic library and the smart card IO library offered by the standard Java Development Kit[7]. This library offers support for communication with smart cards by providing the `javax.smartcardio` package. We used it for the protocol communication with the Java Card smart card on which our client applet was installed.

[1] `https://github.com/pimvullers/sbcred_javacard/`

[2] An implementation of self-blindable credentials for the MULTOS platform is under development at: `https://github.com/pimvullers/sbcred_multos/`

[3] `http://bouncycastle.org/`

[4] `https://github.com/pimvullers/bouncycastle-ext/`

### 3.5.1 Available Elliptic Curve Operations on Java Card

In order to implement the verification protocol on Java Card we need support for elliptic curve cryptography. In this section we summarise the operations available on the Java Card platform and how they can be used to implement the protocol.

#### Elliptic Curve Diffie-Hellman

This functionality is provided by the `javacard.security.KeyAgreement` class. This class can be instantiated using the `getInstance()` method which returns a KeyAgreement object that implements a certain algorithm. The `generateSecret()` method can then be used to actually perform the Diffie-Hellman operation, in our case an elliptic curve multiplication.

The standard Java Card API provides two algorithms [Sun06a]:

`ALG_EC_SVDP_DH`: Elliptic curve secret value derivation primitive, Diffie-Hellman version, as per IEEE P1363.

`ALG_EC_SVDP_DHC`: Elliptic curve secret value derivation primitive, Diffie-Hellman version, with cofactor multiplication, as per IEEE P1363.

Unfortunately, the implementation of these algorithms has two drawbacks:

1. According to the IEEE P1363 standard [IEE00] the shared secret computation by means of ECSVDP-DH and ECSVDP-DHC only returns the $x$-coordinate of the computed point.
2. The Java Card implementation of these algorithms computes the SHA-1 message digest of the output of the derivation primitive to yield a 20 byte result [Sun06a].

Especially this last transformation of the point multiplication result prevents it from being useful for any further computation, other than using it as a secret key.

**JCOP Extension** Luckily the NXP JCOP platform contains some extensions to the standard Java Card API. In particular there is an extension which provides an additional algorithm:

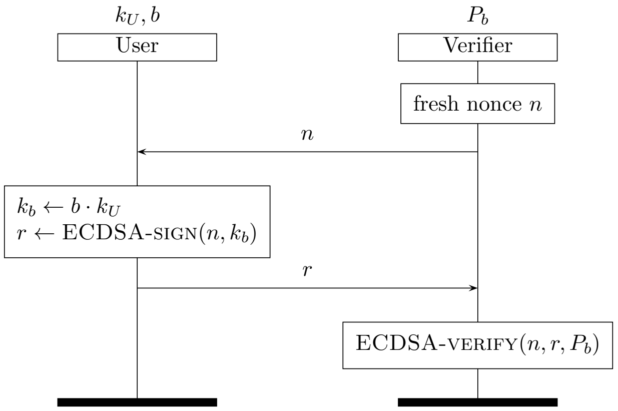
`ALG_EC_SVDP_DH_PLAIN`: the same as `ALG_EC_SVDP_DH` but without SHA-1 post-computation.

This removes the second drawback as mentioned before and only leaves us with the $x$-coordinate of the point multiplication result instead of a point. However, the point can be reconstructed from this coordinate using the elliptic curve formula. By putting in the $x$ value we can compute the corresponding $y$ value. The only unknown in this point reconstruction is the sign of the $y$-coordinate, hence we end up with two candidate points for the multiplication result.

[5] `https://code.google.com/p/bnpairings/`

[6] `https://github.com/pimvullers/sbcred_terminal/`

[7] Since version 6.0.



$k_U, b$
User
$P_b$
Verifier
fresh nonce $n$
$n$
$k_b \leftarrow b \cdot k_U$
$r \leftarrow$ ECDSA-SIGN$(n, k_b)$
$r$
ECDSA-VERIFY$(n, r, P_b)$

Figure 3.4: Proof of possession of $k_b$ using ECDSA.

## 3.5.2 Public Key and Credential Blinding

The first step of the protocol is fairly straightforward. The card has to select the correct credential for the attribute to reveal and blind it together with the public key. As described in Algorithm 3.5 this can be done by calling the Diffie-Hellman operation once for each value. The resulting blinded values $P_b$ and $C_b$ are then returned to the terminal.

**Algorithm 3.5** Public Key and Credential Blinding.

```
1: function SBC-BLIND($b, P_U, C_a$)
2:     $P_b \leftarrow$ GENERATESECRET($P_U, b$)
3:     $C_b \leftarrow$ GENERATESECRET($C_a, b$)
4:     return $P_b, C_b$
```

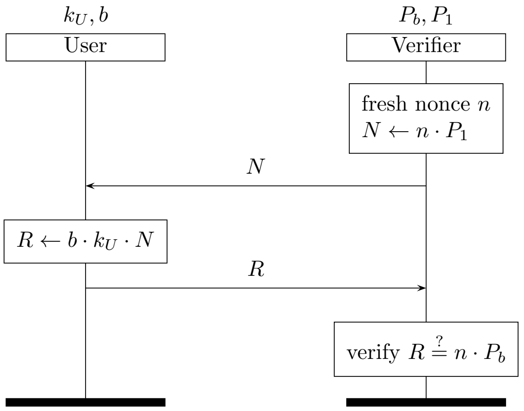
## 3.5.3 Proof of Possession of the Private Key

Of course, when a terminal receives such a pair $P_b$, $C_b$ it should not only check that $C_b$ is a proper signature on $P_b$, but also that the card knows the private key corresponding to the public key $P_b$. This can be done via standard challenge-response exchange, for example using ECDSA.

### Using the ECDSA Signature Scheme

As the ECDSA digital signature algorithm is supported by the Java Card API, it has been our first choice for implementing the challenge-response protocol to establish proof of possession. As input for this algorithm we need the blinded private key $k_b = b \cdot k_U$.

Since modular multiplication is not provided by the Java Card API we used the NatLib library developed by Hendrik Tews [TJ09]. This library implements modular arithmetic on Java Card using basic byte operations as the RSA operations available



$k_U, b$
User
$P_b, P_1$
Verifier
fresh nonce $n$
$N \leftarrow n \cdot P_1$
$N$
$R \leftarrow b \cdot k_U \cdot N$
$R$
verify $R \overset{?}{=} n \cdot P_b$

Figure 3.5: Proof of possession of $k_b$ using minimal Boneh-Lynn-Shacham.

on the platform do not support the short values used in our protocol. The resulting proof of possession protocol is depicted in Figure 3.4 and the implementation is described by Batina et al. [BHJ+10]. However, the NatLib library causes most of the running time of the protocol, as discussed in Section 3.6. This made us look into other alternatives for this protocol, where the use of this library is no longer needed.

#### Using a minimal variant of the Boneh-Lynn-Shacham Signature Scheme

A feasible alternative to the ECDSA scheme is what we call a minimal variant of the Boneh-Lynn-Shacham signature scheme. Since it consists of only a single point multiplication using the private key the workload on the card is minimal. We can also exploit the structure of the message to be signed to reduce the verification algorithm to a single point multiplication as well. The resulting proof of possession protocol is depicted in Figure 3.5.

This approach requires us to compute $b \cdot k_U \cdot N$. This can be split up in two ways, in a modular multiplication and a point multiplication or in two point multiplications. Since the modular multiplication caused a slow-down in the previous approach we perform the point multiplications. For this we exploit the elliptic curve cryptographic key generation functionality[8] provided by the Java Card API. This functionality is used to generate the blinding factor and blind the received challenge at the same time. This allows us to simply sign the blinded challenge with the private key using the Diffie-Hellman operation, as shown in Algorithm 3.6.

### 3.5.4 Terminal Application

The terminal application needs to cope with the shortcomings of the Java Card applet. This comes down to the fact that the terminal has to reconstruct the points $P_b$, $C_b$ and $R$ received from the card before they can be processed any further.

[8]Using the Diffie-Hellman operation twice is not an option due to the first drawback mentioned in Section 3.5.1: the function does not return a point that can be used for another point multiplication directly, and performing the point reconstruction on the card would be to costly.

**Algorithm 3.6** Self-blindable credential verification.

```
1: function SBC-PROVE($P_U, k_U, C_a, N$)
2:     $(b, N_b) \leftarrow$ GENERATEKEY($N$)
3:     $R \leftarrow$ GENERATESECRET($N_b, k_U$)
4:     $P_b \leftarrow$ GENERATESECRET($P_U, b$)
5:     $C_b \leftarrow$ GENERATESECRET($C_a, b$)
6:     return $P_b, C_b, R$
```

If we know the $x$-coordinate of a point on the curve, the square of the corresponding $y$-coordinate is known, namely as $y^2 = x^3 + ax + b$. By taking the square root of $x^3 + ax + b$ we find either $y$ or $-y$.

This reconstruction is a simple guess work, trying different signs for the $y$-coordinates of the points. For the proof of possession verification guessing the sign is not a real issue since this verification is only a single point multiplication, although performing this multiplication twice is of course not optimal. For the pairing signature verification (3.1) simple guessing is not desirable. Therefore we exploit the bilinearity of the pairing to avoid computing more than two pairings, as would be the case without point reconstruction.

First we calculate $e_1 = e(P_b, Q_a)$ and $e_2 = e(C_b, P_2)$ where we take any sign for the $y$-coordinate of $C_b$. If $e_1 = e_2$, which happens if we have two right, or two wrong, signs in the first parameters of the pairing, the verification succeeds. In the remaining case, which means we took one right and one wrong sign, we check whether $e_1 \cdot e_2 = 1$ holds. If it holds, the verification also succeeds. This is true because of the following. If $e_1 \neq e_2$, the error is caused by the wrong sign resulting in one pairing being the inverse of the other, that is, $e_2 = e_1^{-1}$. Here we can use that $e_1 \cdot e_2 = e_1 \cdot e_1^{-1} = 1$ to avoid an extra pairing calculation for the negated point of $C_b$.

## 3.6 Performance Results

For our test we selected three Barreto-Naehrig curves (see Section 2.3.2) for keys of length 128, 160 and 192 bits. The domain parameters $p$ and $n$ for these curves are generated by the Barreto-Naehrig indices $u = 1678770247$, $u = 448873116367$ and $u = 105553250485267$ respectively. The curve $E$ is defined as $y^2 = x^3 + 3$, that is, take $a = 0$ and $b = 3$ in the general form $y^2 = x^3 + ax + b$, with the default generator $P = (1, 2)$.

These key lengths have been chosen to indicate performance for various levels of security, that is, protection against fraud. A key length of 128 bits provides borderline security, whereas 160 and 192 bits provide, respectively, a minimal and a standard level of security [ECR09].

The results of our tests are summarised in Table 3.1. These values are the average of ten test runs for each configuration. The table shows the (accumulated) duration (in milliseconds) of the requests to the card.

When we compare the two implementations it can be seen that the duration of

Table 3.1: Test results for various key lengths.

| key length (bits) | ECDSA (ms) | minimal Boneh-Lynn-Shacham (ms) | communication (bytes) |
|---|---|---|---|
| 192 | 2748 | 787 | 155 |
| 160 | 1860 | 645 | 135 |
| 128 | 1599 | 535 | 115 |

the ECDSA variant is significantly larger than the minimal Boneh-Lynn-Shacham solution. This contrast can be explained by the available support from the cryptographic coprocessor. For the blinding operations in the latter the applet only uses elliptic curve primitives provided by the coprocessor to perform the required point multiplications. The blinding for the ECDSA variant, which requires a modular multiplication, has to be calculated *without* the help of the coprocessor.

In theory it is possible to abuse the RSA cipher (and hence use the coprocessor) to do large part of the modular multiplication by using the fact that $4ab = (a + b)^2 - (a - b)^2$, as in [SGPV09, TJ09]. The squares in this equation can be performed by doing an RSA encryption/exponentiation with a suitable RSA public key, that is, one with the exponent 2 and the required modulus. The numbers $a + b$ and $a - b$ are then just messages to be encrypted using the RSA cipher, which is provided by the Java Card API.

We tried this approach, but unfortunately with no success. The main obstacle is that the RSA cipher on the card operates only within valid bit lengths for RSA keys, starting with 512 bit keys. Although the number to be multiplied (the message) can be any value, the number of non-zero bits in the modulus has to be at least 488 bits for 512 bit keys according to our tests. Since our modulus is only 192 bit long the card refused to perform an RSA encryption with such a short modulus value. However, we believe that a more flexible RSA implementation on the card would allow this optimisation.

To get some more information about how the running time of the improved implementation is spent on the card we measured how long it takes to perform the individual operations. The results of these measurements can be found in Table 3.2. The columns indicate the time needed to perform a single operation. The processing overhead is determined by subtracting one key generation and three key agreements from the protocol duration.

On the one hand, performing a scalar point multiplication (a key agreement operation) is quite efficient, using less then 100 milliseconds for the calculation. On the other hand, performing a scalar point multiplication, combined with generating a random value, (a key generation operation) is disappointing, taking more than a factor three longer than just the multiplication. A possible explanation for this difference is that a different calculation method is used, which might also explain the fact that a key generation can return a complete point, whereas a key agreement can only return the $x$-coordinate.

Table 3.2: Test results for the API primitives.

| key length (bits) | key generation (ms) | key agreement (ms) | processing overhead (ms) |
|---|---|---|---|
| 192 | 379 | 98 | 114 |
| 160 | 307 | 78 | 104 |
| 128 | 242 | 62 | 107 |

A large benefit of our use of elliptic curve cryptography is the small amount of data that needs to be exchanged between the terminal and the card. For key lengths of 192, 160 and 128 bits the total amount of bytes exchanged is 155, 135 and 115 respectively. This would allow an implementation to use a *single* APDU pair (command and response) for all communication. This is in strong contrast with discrete logarithm or RSA-based protocols [SGPV09, TJ09] which already require multiple APDUs to transfer a single command.

When we consider the memory requirements of this implementation a credential only needs 37 bytes of storage for a key length of 128 bits. For a key length of 192 bits this is only 53 bytes to store the issuer's signature and a reference to the attribute statement. To perform the computations, using a 128-bits key, at least 130 bytes of RAM are required to store the session variables, and just 194 bytes for a key length of 192 bits. Taking these requirements into account a single smart card can hold many[9] credentials and has no shortage of RAM for performing the necessary computations. Hence, the main limitation for this technology is the limited support for elliptic curve cryptography on the current generation of smart cards.

[9]Given a card with 64KB of EEPROM and a key length of 192 bits, up to 1200 credentials can be stored.

# Chapter 4

# U-Prove

Stefan Brands provided the first integral description of the U-Prove technology in his thesis [Bra00], after which he founded the company Credentica to implement and sell this technology. Microsoft acquired Credentica and published the U-Prove cryptographic specification [BP10, PZ13] under the Open Specification Promise[1] together with open source reference software development kits in C# and Java.

The U-Prove technology is centred around a so-called U-Prove token. This token serves as a pseudonym for the user. It contains a number of attributes which can be selectively disclosed to a verifier. Hence the user decides which attributes to show and which to withhold (for example, one can reveal the birth date, but not the residence address). Finally there is the token's public-key, which aggregates all information in the token, and a signature from the issuer over this public-key to ensure the authenticity.

A previous attempt to implement this technology on a smart card by Tews and Jacobs [TJ09], based on Brands' description [Bra00], resulted in a highly involved Java Card applet with running times in the order of 5–10 seconds which make it not really usable in practice. They concluded that the lack of cryptographic hardware support through the Java Card API was the main cause of the slow performance. Therefore, we decided to develop our application using the MULTOS platform, which offers a much more suitable API. Our implementation, which we describe in Section 4.5, not only has a much better performance but is also, except from some minimal limitations, compatible with the development kits released by Microsoft.

## 4.1 Schnorr Signature Scheme

The Schnorr signature scheme [Sch89, Sch91] relies on the Schnorr identification protocol [Sch89] which is a special instance of the interactive protocol of Chaum, Evertse and Van de Graaf [CEvdG88] that proves knowledge of a discrete logarithm key pair. It is derived from this identification scheme by replacing the verifier's challenge by a hash value according to the Fiat-Shamir heuristic [FS87] as can be seen in Algorithm 4.1.

[1]`http://www.microsoft.com/interop/osp/`

**Algorithm 4.1** Generate a Schnorr signature.

```
1: function SCHNORR-SIGN($m, (p, q, g), x$)
2:     $u \leftarrow$ RANDOM( )
3:     $a \leftarrow g^u \mod p$
4:     $c \leftarrow$ HASH($a, m$)
5:     $r \leftarrow u + c \cdot x \mod q$
6:     return $(c, r)$
```

**Algorithm 4.2** Verify a Schnorr signature.

```
1: function SCHNORR-VERIFY($(c, r), m, (p, q, g), h$)
2:     $\hat{a} \leftarrow g^r \cdot h^{-c} \mod p$
3:     if $c \neq$ HASH($\hat{a}, m$) then
4:         return INVALID
5:     return VALID
```

The private key is a random value $x$. The corresponding public key is a value $h = g^x \mod p$ together with a description of the prime-order group in which the computations take place. As an example we use $(p, q, g)$, which denotes a subgroup of $\mathbb{Z}_p^*$ of prime-order $q$ with generator $g$. The signature over a message $m$ is the resulting pair $(c, r)$.

Verification of such a Schnorr signature $(c, r)$ starts with the reconstruction of the input value for the hash function based on $r$, according to the following equation:

$$\hat{a} = g^r \cdot h^{-c} = g^{u+c \cdot x} \cdot g^{-c \cdot x} = g^u = a \mod p \tag{4.1}$$

If the signature is valid, that is, this equation holds, the resulting output of the hash function matches $c$ (see Algorithm 4.2).

### 4.1.1 Blind Signatures

Figure 4.1 depicts the protocol for generating blind signatures [PS96]. The steps executed by the signer are similar to the original signing protocol. In Algorithm 4.3 the signer constructs a commitment $a$, to the randomisation value $u$, which is sent to the recipient of the signature.

The recipient generates blinding values $v$ and $w$ to hide the message $m$ that it wants to get signed. A (blinded) commitment $c$ to this message is generated (see Algorithm 4.4) and sent to the signer.

**Algorithm 4.3** Prepare for a blind Schnorr signature.

```
1: function SCHNORR-PREPARE($(p, q, g)$)
2:     $u \leftarrow$ RANDOM( )
3:     $a \leftarrow g^u \mod p$
4:     return $a, u$
```

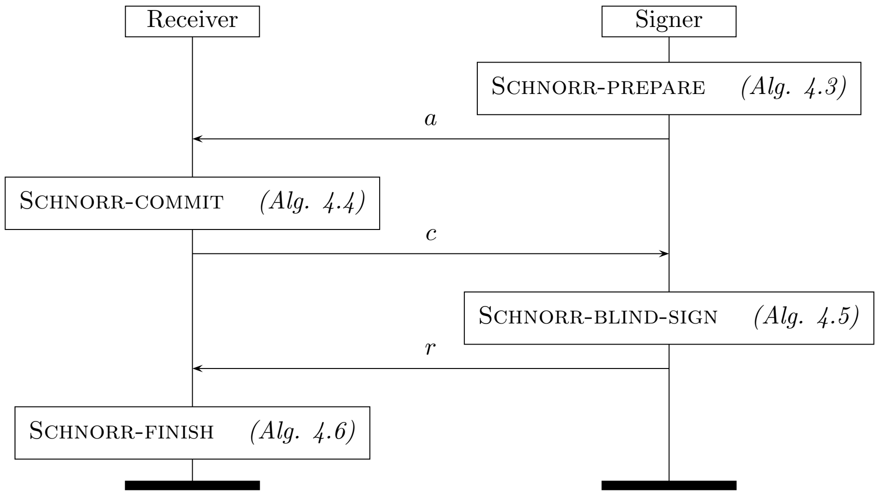

Receiver
Signer
Schnorr-prepare (Alg. 4.3)
a
Schnorr-commit (Alg. 4.4)
c
Schnorr-blind-sign (Alg. 4.5)
r
Schnorr-finish (Alg. 4.6)

Figure 4.1: Protocol for generating blind Schnorr signatures.

**Algorithm 4.4** Commit to the message for a blind Schnorr signature.

```
1: function Schnorr-commit($(m, a, (p, q, g))$)
2:     $v \leftarrow$ Random( )
3:     $w \leftarrow$ Random( )
4:     $a' \leftarrow a \cdot g^v \cdot h^w \mod p$
5:     $c' \leftarrow$ Hash($(a', m)$)
6:     $c \leftarrow c' + w \mod q$
7:     return $c, c', v$
```

Upon receipt of the message commitment, the signer constructs the Schnorr signature value $r$ according to Algorithm 4.5 and returns this value to the recipient. Finally, the recipient checks whether the received value is correct according to (4.1) and computes the last element of the blind Schnorr signature $(c', r')$.

The resulting signature $(c', r')$ can then be verified using the regular verification procedure, Algorithm 4.2. This works since:

$$\begin{aligned}\hat{a} = g^{r'} \cdot h^{-c'} &= g^{r+v} \cdot g^{-c' \cdot x} = g^{u+c \cdot x+v} \cdot g^{-c' \cdot x} = g^{u+(c'+w) \cdot x+v} \cdot g^{-c' \cdot x} \\ &= g^u \cdot g^v \cdot g^{c' \cdot x} \cdot g^{w \cdot x} \cdot g^{-c' \cdot x} = g^u \cdot g^v \cdot g^{x \cdot w} = a \cdot g^v \cdot h^w = a' \mod p\end{aligned}$$

**Algorithm 4.5** Generate a blind Schnorr signature.

```
1: function Schnorr-blind-sign($(c, u, (p, q, g), x)$)
2:     $r \leftarrow u + c \cdot x \mod q$
3:     return $r$
```

**Algorithm 4.6** Finish a blind Schnorr signature.

```
1: function SCHNORR-FINISH($r, a, c, c', v, (p, q, g), h$)
2:     $\hat{a} \leftarrow g^r \cdot h^{-c} \mod p$
3:     if $a \neq \hat{a}$ then
4:         return INVALID
5:     $r' \leftarrow r + v \mod q$
6:     return $(c', r')$
```

## 4.2 U-Prove Credentials

The U-Prove technology is built around U-Prove tokens. These tokens are in principle a collection of attributes signed by an issuer. The issuer's public key $(g_0, \{g_i\}_{i \in \mathcal{A}})$ consists of a value $g_0 = g^x \mod p$ which commits to the private key $x$ and a random generator $g_i$ for each possible attribute ($\mathcal{A}$ denotes the set of attribute indices).

In order for the attributes $\{a_i\}_{i \in \mathcal{A}}$, to be signed they are aggregated into what is called the token public key $h = h'^s \mod p$ where $s$ is the user's secret and $h'$ is the aggregation of the attributes using the issuer's public key:

$$h' = g_0 \cdot \textstyle\prod_{i \in \mathcal{A}} g_i^{a_i} \mod p \tag{4.2}$$

The corresponding token private key $s' = s^{-1} \mod q$. Together with the issuer's signature $(z', c', r')$, generated during the issuance process (see Section 4.3), these values form a U-Prove token. Such a token serves as a pseudonym for the user. A collection of U-Prove tokens, where only the signatures differ, can be seen as a U-Prove credential.

A U-Prove credential usually consist of multiple tokens to achieve anonymity, since a single token acts as a pseudonym for the user. This means that whenever the user does not want to be traced during a credential verification (see Section 4.4) she should use a fresh token to prevent the verifier from linking the token to previous transactions.

To summarise, a U-Prove credential, as depicted in Figure 4.2, consists of:

- a collection of attributes $\{a_i\}_{i \in \mathcal{A}}$ aggregated in $h'$,
- a key pair ($h = h'^s \mod p$, $s' = s^{-1} \mod q$), based on the user's secret $s$, and
- a number of signatures $(z', c', r')$, over the token public key (and hence over the user's secret and the attributes).

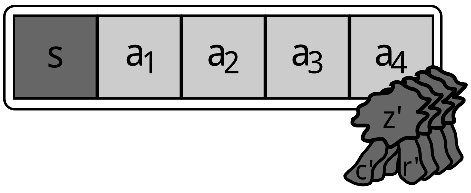


Figure 4.2: A visual representation of a U-Prove credential.

## 4.3 Credential Issuance

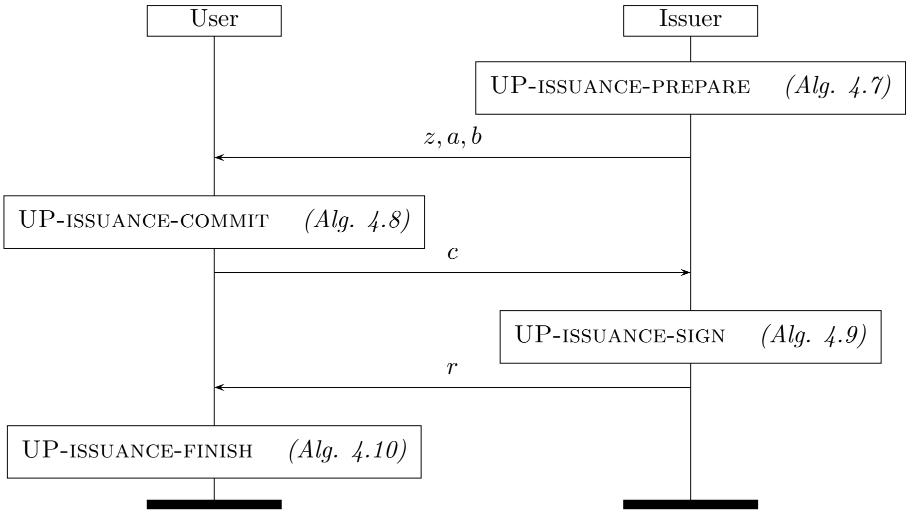

User
Issuer
UP-issuance-prepare (Alg. 4.7)
$z, a, b$
UP-issuance-commit (Alg. 4.8)
$c$
UP-issuance-sign (Alg. 4.9)
$r$
UP-issuance-finish (Alg. 4.10)

Figure 4.3: Protocol for U-Prove credential issuance.

Figure 4.3 depicts the issuance protocol for a U-Prove token. The issuer starts by computing $z = h'^x \mod p$ which combines its private key $x$ with the attributes $a_{i \in \mathcal{A}}$, aggregated according to (4.2). Next, the issuer commits to the randomisation value $u$ using both the generator $g$ and the aggregated attributes $h'$ that are going to be signed (see Algorithm 4.7).

**Algorithm 4.7** Prepare for U-Prove issuance.

```
1: function UP-issuance-prepare($h', (p, q, g), x$)
2:     $z \leftarrow h'^x \mod p$
3:     $u \leftarrow$ Random( )
4:     $a \leftarrow g^u \mod p$
5:     $b \leftarrow h'^u \mod p$
6:     return $z, a, b, u$
```

The user starts in Algorithm 4.8 by generating a fresh secret $s$ and computes the token private key $s'$. She then combines this secret with the aggregated attributes $h'$ and the value $z$, received from the issuer, to obtain, respectively, the token public key and the first element of the signature $z'$. The remaining steps compute the commitment to these values similar to Schnorr's blind signature scheme (Algorithm 4.4).

The issuer then performs the next step of Schnorr's blind signature scheme (Algorithm 4.5) and signs the commitment $c$ received from the user according to Algorithm 4.9.

Finally, in Algorithm 4.10, the user computes $r'$, the last element of the token signature $(z', c', r')$. Using this value she can also verify whether the values received

**Algorithm 4.8** Commit to the attributes for U-Prove issuance.

```
1: function UP-ISSUANCE-COMMIT($h', z, a, b, (p, q, g)$)
2:     $s \leftarrow$ RANDOM( )
3:     $s' \leftarrow s^{-1} \mod q$
4:     $h \leftarrow h'^{s} \mod p$
5:     $z' \leftarrow z^{s} \mod p$
6:     $v \leftarrow$ RANDOM( )
7:     $w \leftarrow$ RANDOM( )
8:     $a' \leftarrow a \cdot g^{v} \cdot g_0^{w} \mod p$
9:     $b' \leftarrow b^{s} \cdot h^{v} \cdot z'^{w} \mod p$
10:    $c' \leftarrow$ HASH($h, z', a', b'$) $\mod q$
11:    $c \leftarrow c' + w \mod q$
12:    return $z', a', b', c, c', v, s'$
```

**Algorithm 4.9** Sign the attributes for U-Prove issuance.

```
1: function UP-ISSUANCE-BLIND-SIGN($c, u, (p, q, g), x$)
2:     $r \leftarrow u + c \cdot x \mod q$
3:     return $r$
```

from the issuer are correct:

$$\begin{aligned}
a' \cdot b' &= a' \cdot b' \cdot g_0^{c'} \cdot g_0^{-c'} \cdot h^{c' \cdot x} \cdot h^{-c' \cdot x} \\
&= a \cdot g^{v} \cdot g_0^{w} \cdot b^{s} \cdot h^{v} \cdot z'^{w} \cdot g_0^{c'} \cdot g_0^{-c'} \cdot h^{c' \cdot x} \cdot h^{(-c+w) \cdot x} \\
&= g^{u} \cdot g^{v} \cdot g_0^{w} \cdot h'^{u \cdot s} \cdot h^{v} \cdot z^{s \cdot w} \cdot g_0^{c'} \cdot g_0^{-c'} \cdot h^{c' \cdot x} \cdot h^{-c \cdot x} \cdot h^{w \cdot x} \\
&= g^{u} \cdot g^{v} \cdot g^{w \cdot x} \cdot h^{u} \cdot h^{v} \cdot z^{w \cdot s} \cdot g^{c' \cdot x} \cdot g_0^{-c'} \cdot h^{c' \cdot x} \cdot h^{w \cdot x} \cdot z^{-c \cdot s} \\
&= g^{u} \cdot g^{c' \cdot x} \cdot g^{w \cdot x} \cdot g^{v} \cdot h^{u} \cdot h^{c' \cdot x} \cdot h^{w \cdot x} \cdot h^{v} \cdot g_0^{-c'} \cdot z^{-c \cdot s} \cdot z^{w \cdot s} \\
&= g^{u + c' \cdot x + w \cdot x + v} \cdot h^{u + c' \cdot x + w \cdot x + v} \cdot g_0^{-c'} \cdot z^{(-c+w) \cdot s} \\
&= (g \cdot h)^{u + c' \cdot x + w \cdot x + v} \cdot g_0^{-c'} \cdot z'^{-c+w} \\
&= (g \cdot h)^{u + c \cdot x + v} \cdot g_0^{-c'} \cdot z'^{-c+w} \\
&= (g \cdot h)^{r+v} \cdot g_0^{-c'} \cdot z'^{-c'} \\
&= (g \cdot h)^{r'} \cdot (g_0 \cdot z')^{-c'} \mod p
\end{aligned}$$

## 4.4 Credential Verification

The verification of a U-Prove credential consists of two parts. First, the verifier must check whether the signature over the token public key is valid. To this end the user sends the token public key and signature to the verifier as can be seen in Figure 4.3.

**Algorithm 4.10** Finish U-Prove issuance.

```
1: function UP-ISSUANCE-FINISH($a', c', z', v, r, (p, q, g), h$)
2:     $r' \leftarrow r + v \mod q$
3:     if $a' \cdot b' \neq (g \cdot h)^{r'} \cdot (g_0 \cdot z')^{-c'}$ then
4:         return INVALID
5:     return $(z', c', r')$
```

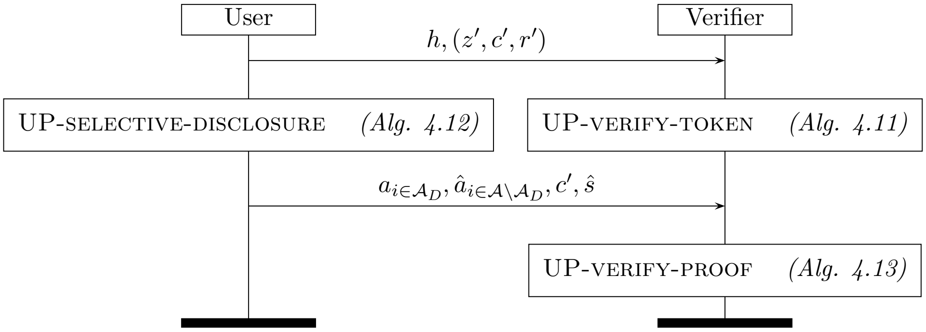

User
Verifier
$h, (z', c', r')$
UP-SELECTIVE-DISCLOSURE (Alg. 4.12)
UP-VERIFY-TOKEN (Alg. 4.11)
$a_{i \in \mathcal{A}_D}, \hat{a}_{i \in \mathcal{A} \setminus \mathcal{A}_D}, c', \hat{s}$
UP-VERIFY-PROOF (Alg. 4.13)

Figure 4.4: Protocol for U-Prove credential verification.

The verifier can then verify the validity of the signature $(z', c', r')$ using Algorithm 4.11. Similar to the Schnorr signature verification, the verifier reconstructs the input values $\hat{a}$ and $\hat{b}$ for the hash function according to the following equations:

$$\begin{aligned}\hat{a} &= g^{r'} \cdot g_0^{-c'} = g^{r+v} \cdot g^{-c' \cdot x} = g^{u+c \cdot x+v} \cdot g^{-c' \cdot x} = g^{u+(c'+w) \cdot x+v} \cdot g^{-c' \cdot x} \\ &= g^u \cdot g^{c' \cdot x} \cdot g^{w \cdot x} \cdot g^v \cdot g^{-c' \cdot x} = g^u \cdot g^{w \cdot x} \cdot g^v = g^u \cdot g^v \cdot g_0^w = a \cdot g^v \cdot g_0^w \\ &= a' \mod p\end{aligned}$$

$$\begin{aligned}\hat{b} &= h^{r'} \cdot z'^{-c'} = h^{u+c \cdot x+v} \cdot z^{-c' \cdot s} = h^{u+(c'+w) \cdot x+v} \cdot h'^{-c' \cdot s \cdot x} \\ &= h^u \cdot h^{c' \cdot x} \cdot h^{w \cdot x} \cdot h^v \cdot h^{-c' \cdot x} = h^u \cdot h^v \cdot h'^{s \cdot w \cdot x} = h'^{u \cdot s} \cdot h^v \cdot z^{s \cdot w} = b^s \cdot h^v \cdot z'^w \\ &= b' \mod p\end{aligned}$$

**Algorithm 4.11** Verify a U-Prove token signature.

```
1: function UP-VERIFY-TOKEN($h, (z', c', r'), (p, q, g)$)
2:     $\hat{a} \leftarrow g^{r'} \cdot g_0^{-c'} \mod p$
3:     $\hat{b} \leftarrow h^{r'} \cdot z'^{-c'} \mod q$
4:     if $c' \neq$ HASH$(h, z', \hat{a}, \hat{b})$ then
5:         return INVALID
6:     return VALID
```

If the signature is valid, that is, both equations hold, the resulting output of the hash function matches $c'$.

The second part of the credential verification protocol of Figure 4.4 consists of selective disclosure of the attributes and a proof of knowledge of the token private key. In this part the user sends the disclosed attributes $\{a_i\}_{i \in \mathcal{A}_D}$, where $\mathcal{A}_D$ is the set of disclosed attribute indices, to the verifier and includes the undisclosed attributes in the proof of knowledge of the token private key. This proof of knowledge can be generated using Algorithm 4.12. The freshness of this proof is guaranteed by a nonce $n_D$ provided by the verifier.

**Algorithm 4.12** U-Prove selective disclosure.

```
1: function UP-SELECTIVE-DISCLOSURE($\{a_i\}_{i \in \mathcal{A}}, h, (z', c', r'), s', n_D, (p, q, g)$)
2:     $\tilde{s} \leftarrow$ RANDOM( )
3:     $a \leftarrow h^{\tilde{s}} \mod p$
4:     for each $i \in \mathcal{A} \setminus \mathcal{A}_D$ do
5:         $\tilde{a}_i \leftarrow$ RANDOM( )
6:         $a \leftarrow a \cdot g_i^{\tilde{a}_i} \mod p$
7:     $c' \leftarrow$ HASH($a$)
8:     $c \leftarrow$ HASH($c, a_{i \in \mathcal{A}_D}, n_D$) $\mod q$
9:     $\hat{s} \leftarrow \tilde{s} + c \cdot s' \mod q$
10:    for each $i \in \mathcal{A} \setminus \mathcal{A}_D$ do
11:        $\hat{a}_i \leftarrow \tilde{a}_i - c \cdot a_i \mod q$
12:    return $\{a_i\}_{i \in \mathcal{A}_D}, \{\hat{a}_i\}_{i \in \mathcal{A} \setminus \mathcal{A}_D}, c', \hat{s}$
```

**Algorithm 4.13** U-Prove proof verification.

```
1: function UP-VERIFY-PROOF($\{a_i\}_{i \in \mathcal{A}_D}, \{\hat{a}_i\}_{i \in \mathcal{A} \setminus \mathcal{A}_D}, c', \hat{s}, n_D, (p, q, g)$)
2:     $c \leftarrow$ HASH($c', a_{i \in \mathcal{A}_D}, n_D$)
3:     $\hat{a} \leftarrow g_0^{-c} \cdot h^{\hat{s}} \mod p$
4:     for each $i \in \mathcal{A}_D$ do
5:         $\hat{a} \leftarrow \hat{a} \cdot g_i^{-c \cdot a_i} \mod p$
6:     for each $i \in \mathcal{A} \setminus \mathcal{A}_D$ do
7:         $\hat{a} \leftarrow \hat{a} \cdot g_i^{\hat{a}_i} \mod p$
8:     if $c' \neq$ HASH($\hat{a}$) then
9:         return INVALID
10:    return VALID
```

The selective disclosure proof can be verified using Algorithm 4.13. The verification succeeds if the verifier can successfully reconstruct the commitment $a$ generated by the user. This reconstruction works because of the following equation:

$$\begin{aligned}
\hat{a} &= g_0^{-c} \cdot h^{\hat{s}} \cdot \textstyle\prod_{i\in\mathcal{A}_D} g_i^{-c\cdot a_i} \cdot \prod_{i\in\mathcal{A}\setminus\mathcal{A}_D} g_i^{\hat{a}_i} \\
&= g_0^{-c} \cdot h^{\tilde{s}+c\cdot s'} \cdot \textstyle\prod_{i\in\mathcal{A}_D} g_i^{-c\cdot a_i} \cdot \prod_{i\in\mathcal{A}\setminus\mathcal{A}_D} g_i^{\tilde{a}_i-c\cdot a_i} \\
&= g_0^{-c} \cdot h^{\tilde{s}} \cdot h^{c\cdot s'} \cdot \textstyle\prod_{i\in\mathcal{A}_D} g_i^{-c\cdot a_i} \cdot \prod_{i\in\mathcal{A}\setminus\mathcal{A}_D} g_i^{\tilde{a}_i} \cdot \prod_{i\in\mathcal{A}\setminus\mathcal{A}_D} g_i^{-c\cdot a_i} \\
&= g_0^{-c} \cdot h^{\tilde{s}} \cdot h'^{c\cdot s'\cdot s} \cdot \textstyle\prod_{i\in\mathcal{A}} g_i^{-c\cdot a_i} \cdot \prod_{i\in\mathcal{A}\setminus\mathcal{A}_D} g_i^{\tilde{a}_i} \\
&= h^{\tilde{s}} \cdot h'^{c} \cdot h'^{-c} \cdot \textstyle\prod_{i\in\mathcal{A}\setminus\mathcal{A}_D} g_i^{\tilde{a}_i} \\
&= h^{\tilde{s}} \cdot \textstyle\prod_{i\in\mathcal{A}\setminus\mathcal{A}_D} g_i^{\tilde{a}_i} \\
&= a \mod p
\end{aligned}$$

## 4.5 U-Prove on Smart Cards

The use of U-Prove in combination with a smart card was already envisioned by Brands [Bra00] and published by Microsoft in version 1.1 of the U-Prove cryptographic specification [Paq11a]. Their idea is to use a smart card (or even any trusted computing device) as a manner of protecting U-Prove tokens, which they then call device-protected tokens. This is achieved by having the device contribute one attribute to the token. The actual value of this attribute is, like a private key, only known by the device and will always be hidden. Therefore the device is required during the verification protocol, since the user has to prove knowledge of *all* undisclosed attributes.

Besides adding an additional layer of protection, the U-Prove technology overview [Paq11c] describes a number of other benefits gained when using device-protected tokens. For example, a device can be used to enforce dynamic policies or prevent the use of a token at a blacklisted website. It also helps to enforce non-transferability of tokens by having the user authenticate to the device before allowing it to be used in a transaction. Another option, especially interesting for smart cards, is to use the device as a carrier, or secure roaming store, for entire U-Prove credentials and not just one attribute. This way the U-Prove credentials are always available when needed.

This last feature of a device-protected U-Prove token has one major drawback, namely one will need to trust the device that is used to perform the proving protocol. This is because the actual attribute values are used during the computation steps of this protocol (see Section 4.4). Hence the device must release all information, except its own special attribute, during a protocol run. When using a personal computer this might be acceptable, but in scenarios where the device should be used directly with a verifier, for example at a public transport gate, or at a vending machine for cigarettes, this turns out to be problematic. Since these are the areas of use which are most interesting for us, we decided to develop an implementation which provides the full verification protocol on a smart card instead of using Microsoft's more limited approach.

### 4.5.1 Smart Card Implementation

A very general view of our implementation of the U-Prove technology is that it provides storage for preloaded (e.g. cryptographic domain parameters) and calculated (e.g. generated keys) values of the protocols, as well as attribute storage, and, more importantly, a sequence of hash and modulo prime arithmetic operations to execute the corresponding stages of the protocols. These arithmetic operations are the core of the performance considerations of our implementation. A few hashing operations are executed and multiple exponents over numbers in a large prime field have to be calculated during a verification protocol run.

Considering the size of the U-Prove data that is used in the protocols and the requirements of the MULTOS cryptographic routines (all data for a cryptographic operation needs to be in one continuous array) the first thing to take care of is a careful split of the card data between EEPROM and RAM. Only 960 *bytes* of RAM are available on our development cards, compared to 36 *kilo*bytes of EEPROM. The most frequent use case of the card is the execution of the proving protocol, hence this is where good use of RAM is highly desirable. For that we limited the maximum number of stored attributes to 5 and then we ensured that all variables used in the verification protocol are allocated in RAM.

The initialisation and issuance protocol require more scratch-pad memory than the available RAM, hence we were forced to use EEPROM there. Moreover, the issuance protocol makes use of EEPROM for permanent storage of the issued U-Prove token and other permanent protocol parameters (prime numbers $p$, $q$, etc.). The use of EEPROM for computations has an impact on the running time for these operations, as can be seen in Section 4.6, but this is acceptable given that these operations are normally only used a limited number of times.

This completes the efficiency considerations for our implementation. Otherwise the implementation of the U-Prove protocols is rather straightforward in the MULTOS environment and mostly entails direct calls to the MULTOS API.

### 4.5.2 Integration into the Microsoft U-Prove SDK

The previous section described the implementation of the U-Prove protocols which mainly concerns storage and the mathematical computations. This is, however, not sufficient to use it in combination with Microsoft's U-Prove software development kit. We need to bridge between the high-level Java interfaces defined in this development kit and the low-level APDU interface of the smart card.

We designed the low-level APDU interface to be as simple as possible. Essentially it has to provide three types of functionality:

1. sending data to the card,

2. ask the card to perform the necessary computations, and

3. retrieve the results from the card.

The second type of the interface functionality is easiest, we just defined an APDU instruction for each of the steps in the protocols. For transferring data to and from the card we restricted the values to the maximum amount of data that can be transferred in one APDU (255 bytes). This allows us to just define one APDU instruction per variable, parametrised only with the index if needed (for example $g_i$), for setting or getting a value.

Finally we need to bind this low level APDU API to the interfaces and data types provided by the U-Prove software development kit. Luckily the development kit just uses byte arrays for the external access to the data types such that no additional conversion is needed. The only thing that needs to be done for a data type, for example `IssuerParameters`, is that the setter and getter have to be divided into the individual APDU instructions, for example the `setPublicKey` and `setEncodingBytes` instructions.

All this functionality has been combined into a single Java class which provides setters and getters for the data stored on the card as well as methods for the protocol steps. Using the Java built-in smart card library it serves as an interface between our MULTOS implementation and the U-Prove software development kit.

## 4.6 Performance Results

The two most important factors for us to test in our U-Prove implementation were correctness of the protocol calculations (obviously) and the speed. Testing the correctness was fairly easy. Since we interfaced our card to Microsoft's U-Prove SDK we could simply test it by invoking the protocol runs from the SDK and check the results. During the first stages of the development, partial protocol calculations were verified with the test vectors provided with the U-Prove SDK [Paq11b]. In the whole process a few corner case problems with our calculations surfaced that required minor corrections.

For the performance tests we are restricted by the limits of our MULTOS implementation platform. Namely, on our development cards we are limited to a modulus size of 1024 bits for modular arithmetic,[2] and SHA-1 is the only built-in hashing algorithm available. Although this may sound restrictive, it also makes the choice of the U-Prove protocol configuration (protocol parameters) for our tests easy. We have simply chosen to use the domain parameters fixed to the same ones as in the default configuration of the official U-Prove SDK reference implementation and official U-Prove test vectors [Paq11b], that is 1024 bits for modulus size and SHA-1 for hashing to match with the capabilities of the card.

As we stated in the previous section, for speed we concentrated our implementation efforts on the every day use case of the application, that is, the credential verification protocol. However, we also strived to optimise the rest of the protocols to maintain speed also during the initialisation and issuance parts. For the performance analysis, we executed a number of full protocol runs (initialisation, issuance, proving) on the card in various configurations. First of all we varied the number of stored attributes on the card, then within this attribute range we varied the number of (un)disclosed attributes. As shown in Figure 4.5 this resulted in a running time of 3.6 and 5.5 seconds for the issuance of a U-Prove token with respectively 2 and 5 attributes. The dark grey area on the graph indicates the core running time of the

[2]The card actually supports up to 2048 bits, but then during exponentiation only small enough exponents can be used, a requirement which the U-Prove operations do not satisfy.

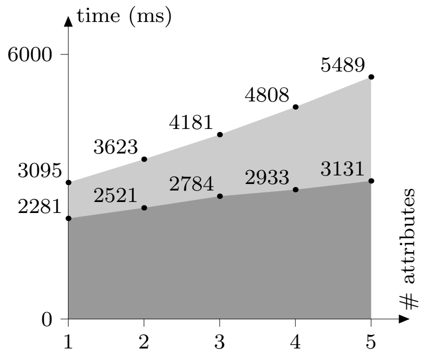


Figure 4.5: U-Prove credential issuance times for various numbers of attributes (▨: computation time, ▨: overhead).

protocol calculations on the card, whereas light grey indicates the remaining overhead. This overhead consists of transferring data to the card and communicating the results of the protocol run between the card and PC.

Correspondingly, the cumulative results for the attribute proving protocol are shown in Figure 4.6. What can be seen in these graphs is that under "full load" our implementation executes the complete proving protocol in just under 0.9 seconds (Figure 4.6d). In this worst-case scenario 5 attributes are stored on the card, none of which are disclosed during the protocol run. In other words, the U-Prove token is only validated for its authenticity without revealing any attributes. Such a scenario is not very likely to occur in reality. In a more likely scenario at least one or two attributes are going to be disclosed and we can also assume that a U-Prove token will contain less attributes (or, that a large number of attributes can be split into several separate U-Prove tokens). The graphs show that reducing the number of stored attributes improves the running time at a rate of 100 milliseconds per attribute. Furthermore, the performance increases along with increasing the number of disclosed attributes, roughly 50 milliseconds per each extra disclosed attribute. Overall, this brings the total execution time for a two attribute token disclosing one attribute to under 0.5 seconds (Figure 4.6a).

One of the reasons to justify Microsoft's device protected approach as described in Section 4.5 is possible resource issues with smart cards (limited storage space and limited speed). Our performance results undermine this argument. The worst case execution time of the proving part is 869 milliseconds. This not only makes the card implementation fast enough to be usable in general, it also makes it usable for "field" applications, for example dispensing machines. Even more, for smaller numbers of attributes the running times become almost acceptable for use in public transport/e-ticketing, where the commonly required card transaction times should stay below 350 milliseconds. We also see a potential to improve the running times using faster smart card hardware, we elaborate on this in the upcoming section. Overall, these good results strongly justify the idea to use U-Prove standalone on a smart card rather than to use Microsoft's device-protected token approach, which has no obvious functional or performance advantages over our approach.

Our implementation requires 1KB to store a credential containing two attributes.

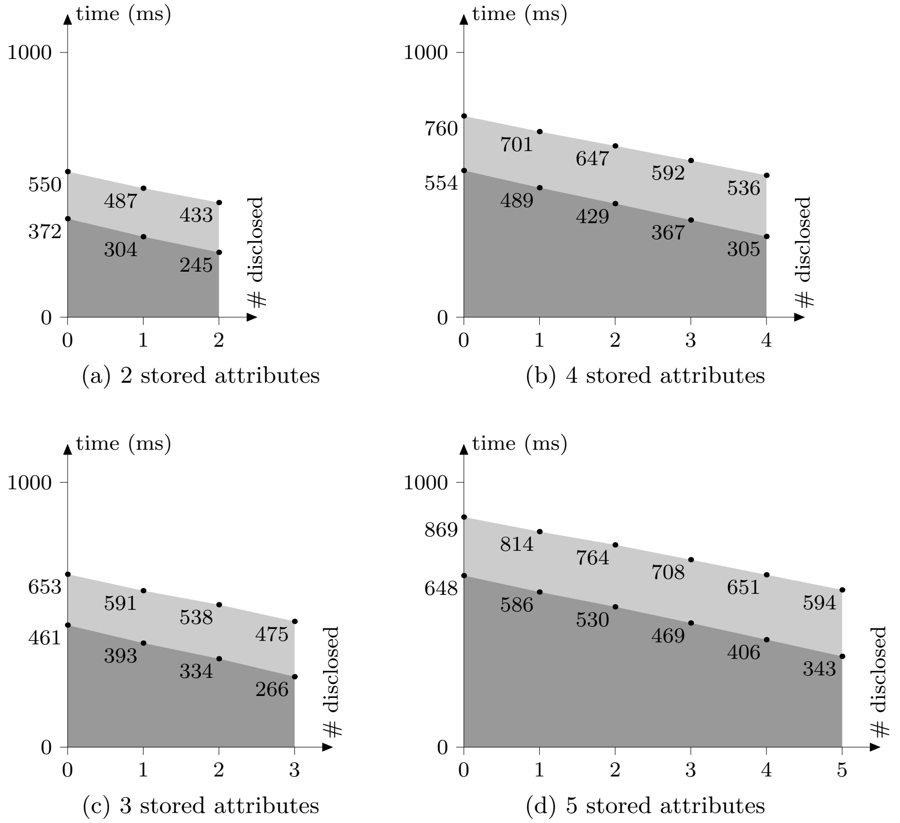


Figure 4.6: U-Prove credential verification times for different configurations (■: computation time, □: overhead).

For each additional attribute another 148 bytes of storage are required. This is of course more than the 148 bytes necessary for the device-protected approach, but this allows for full independence of the terminal. It does, however, become an issue if many credentials have to be stored in order to achieve unlinkability, since a card can typically only hold 50 of such credentials.

When we consider the run-time memory requirements of our implementation the main restrictions become clear. For a credential with 2 attributes 540 bytes of RAM are required for the session variables. An additional 40 bytes are required for each additional attribute in a credential. These values are based on a modulus of 1024 bits and attribute values of 160 bits (the SHA-1 hashed representation of the actual attribute values).

Based on these results we can conclude that the limitations of our implementation are imposed by the limited resources of the MULTOS smart card. We had to limit the prime modulus size to 1024 bits, use only SHA-1 hashing, and because of the available RAM (less than 1KB) on the card we could only allow for the maximum of 5 attributes, each one up to 255 bytes in size. Otherwise our implementation is fully flexible and provides full U-Prove functionality, *including* the smart card features described in Section 4.5. However, it is not uncommon for modern smart cards to support up to 2048 bits for modulus size and 2 kilobytes of RAM, only no such MULTOS cards were available to us.

An alternative would be to build a U-Prove implementation using elliptic curve cryptography, as this is supported by the technology. This would drastically reduce the communication overhead as well as the memory requirements. Unfortunately, as mentioned in the previous chapter, support for elliptic curves on smart cards is limited, which prevents the development of an efficient elliptic curve-based implementation for now.

# Chapter 5

# Identity Mixer

Identity Mixer [CL01, IBM12] (also known as Idemix) is an attribute-based credential system, developed at IBM Research in Zürich, that enables strong authentication and privacy at the same time. The first prototype [CvH02] was developed in 2002 and has been improved over the years [BCC04b, CG05]. An open source Java implementation[1] was released in 2010 as part of IBM's open innovation initiative[2].

The core of the Identity Mixer technology is the Camenisch-Lysyanskaya signature scheme [CL03, Lys02]. This scheme is an ideal building block for privacy-preserving technologies as Camenisch and Lysyanskaya provide protocols for

1. issuing signatures on committed values (such that the signer has no information about the signed value), that is, blind signatures [Cha83], and
2. proving knowledge of a signature on a committed value.

Furthermore this scheme supports multiple messages to be signed at once. This allows a number of attributes to be combined into a single credential. In contrast to the U-Prove technology, an Identity Mixer credential can be randomised (like the self-blindable credentials) allowing it to be used multiple times while remaining anonymous. This is also the only technology that offers provable security. This means that all protocols have been proven formally to be secure.

In this chapter we focus on the core features of the Identity Mixer system: credential issuance and selective disclosure of attributes. The technology offers many more features that might be interesting to users of an attribute-based credential system, but we focus on what can currently be achieved on a smart card with an acceptable performance [VA13]. Additional options can still be added, but at the cost of an increased transaction time.

## 5.1 Camenisch-Lysyanskaya Signature Scheme

This description of the Camenisch-Lysyanskaya signature scheme [CL03, Lys02] is based on the direct anonymous attestation explanation by Camenisch [Cam07] and

[1]`http://prime.inf.tu-dresden.de/idemix/`
[2]`http://www.zurich.ibm.com/news/10/innovation.html`

the specification of the Identity Mixer cryptographic library [IBM12]. These documents include the efficiency improvements which have been presented by Brickell, Camenisch, and Chen [BCC04b] and by Camenisch and Groth [CG05].

### 5.1.1 Keys

The private key in this scheme comprises two Sophie Germain primes $p'$ and $q'$. The public key consists of a special RSA modulus $n = p \cdot q$ where $p = 2 \cdot p' + 1$ and $q = 2 \cdot q' + 1$ are safe primes. Furthermore we need the following parameters from $QR_n = \{x \in \mathbb{Z}_n : y \in Z_n \wedge x = y^2 \mod n\}$, the group of quadratic residues modulo $n$:

- a generator $S \in QR_n$, with order $p' \cdot q'$, that generates $\langle S \rangle$, the subgroup of $QR_n$ in which all computations take place;
- a base $R_i = S^{x_i} \mod n$, where $x_i \in_R [2, p' \cdot q' - 1]$, for each message $m_i$ to be signed using this key; and
- an auxiliary value $Z = S^{x_z} \mod n$, where $x_z \in_R [2, p' \cdot q' - 1]$, such that all computations remain within $\langle S \rangle$.

Hence the resulting public key is $(n, S, Z, \{R_i\}_{i \in \mathcal{M}})$ where $\mathcal{M}$ denotes the set of message indices, and hence the maximum number of messages, supported by this key.

In order to guarantee that such a public key has been constructed correctly, that is, all values are elements of $\langle S \rangle$, a proof of correctness can be constructed (Algorithm 5.1, based on [BCC04b, Appendix A]):

**Algorithm 5.1** Proof correctness of a Camenisch-Lysyanskaya public key.

```
function CL-PROVE-KEY($(n, S, Z, \{R_i\}_{i \in \mathcal{M}})$)
    for each $j \in \mathcal{H}$ do
        $u_j \leftarrow$ RANDOM( )
        $Z'_j = S^{u_j} \mod n$
        for each $i \in \mathcal{M}$ do
            $v_{(i,j)} \leftarrow$ RANDOM( )
            $R'_{(i,j)} = S^{v_{(i,j)}} \mod n$
    $c \leftarrow$ HASH$(n, S, Z, \{R_i\}_{i \in \mathcal{M}}, (Z'_j)_{j \in \mathcal{H}}, \{R'_{(i,j)}\}_{i \in \mathcal{M}, j \in \mathcal{H}})$
    for each $j \in \mathcal{H}$ do
        $r_j = u_j - c_j \cdot x_z \mod (p' \cdot q')$
        for each $i \in \mathcal{M}$ do
            $s_{(i,j)} \leftarrow v_{(i,j)} - c_j \cdot x_i \mod (p' \cdot q')$
    return $(c, \{r_j\}_{j \in \mathcal{H}}, \{s_{(i,j)}\}_{i \in \mathcal{M}, j \in \mathcal{H}})$
```

$$SPK\{(\alpha_z, \{\alpha_i\}_{i\in\mathcal{M}}) : Z = S^{\alpha_z} \mod n \wedge \\ \forall_{i\in\mathcal{M}} R_i = S^{\alpha_i} \mod n\}(n, S, Z, \{R_i\}_{i\in\mathcal{M}})$$

This proof uses binary challenges, hence we need to generate commitments and responses for all bits of the challenge $\{c_j\}_{j\in\mathcal{H}}$, where $\mathcal{H}$ denotes the set of bits generated by the HASH function. These binary challenges are necessary to satisfy the special soundness property of the security proof [Cam07, Section 3.5]. Normally, one could just use the challenge $c$ in combination with the strong RSA assumption to satisfy this property, but in this case the strong RSA assumption does not hold since the party that generated the key knows the factorisation of $n$.

The proof can be verified using Algorithm 5.2, which reconstructs the input of the hash based on the proof values and then checks whether the output of the hash matches the given value. These values can be reconstructed because:

$$Z^{c_j} \cdot S^{r_j} = (S^{x_z})^{c_j} \cdot S^{r_j} = (S^{x_z})^{c_j} \cdot S^{u_j - c_j x_z} = S^{u_j} = Z'_j \mod n, \text{ and}$$

$$R_i^{c_j} \cdot S^{s(i,j)} = (S^{x_i})^{c_j} \cdot S^{s(i,j)} = (S^{x_i})^{c_j} \cdot S^{v(i,j) - c_j x_i} = S^{v(i,j)} = R'_{(i,j)} \mod n.$$

**Algorithm 5.2** Verify correctness of a Camenisch-Lysyanskaya public key.

```
1: function CL-VERIFY-KEY($(n, S, Z, \{R_i\}_{i\in\mathcal{M}}), (c, \{r_j\}_{j\in\mathcal{H}}, \{s_{(i,j)}\}_{i\in\mathcal{M}, j\in\mathcal{H}})$)
2:     for each $j \in \mathcal{H}$ do
3:         $Z'_j \leftarrow Z^{c_j} \cdot S^{r_j} \mod n$
4:         for each $i \in \mathcal{M}$ do
5:             $R'_{(i,j)} \leftarrow R_i^{c_j} \cdot S^{r(i,j)} \mod n$
6:     if $c \neq$ HASH($n, S, Z, \{R_i\}_{i\in\mathcal{M}}, Z'_{j\in\mathcal{H}}, \{R'_{(i,j)}\}_{i\in\mathcal{M}, j\in\mathcal{H}}$) then
7:         return INVALID
8:     return VALID
```

### 5.1.2 Basic Signature Scheme

In order to sign a collection of messages $\{m_i\}_{i\in\mathcal{M}}$, these $m_i$ first have to be aggregated into a single group element $Q$ according to the following equation:

$$Q = \frac{Z}{S^v \cdot \prod_{i\in\mathcal{M}} R_i^{m_i}} \mod n, \tag{5.1}$$

where $v$ is a random number. This value $v$ is used in Section 5.1.3 for blinding the messages that have to remain hidden, and in Section 5.1.4 to randomise the signature.

The actual signature generation process is similar to the RSA signature scheme. The first step is the generation of a random prime $e$ which is used as the ephemeral RSA public key for this signature. Next, the RSA private key $d = e^{-1} \mod (p' \cdot q')$

**Algorithm 5.3** Generate a basic Camenisch-Lysyanskaya signature.

```
function CL-SIGN($\{m_i\}_{i \in \mathcal{M}}, (n, S, Z, \{R_i\}_{i \in \mathcal{M}}), (p', q')$)
    $v \leftarrow$ RANDOM( )
    $U \leftarrow S^v \mod n$
    for each $i \in \mathcal{M}$ do
        $U \leftarrow U \cdot R_i^{m_i} \mod n$
    $Q \leftarrow Z \cdot U^{-1} \mod n$
    $e \leftarrow$ RANDOMPRIME( )
    $d \leftarrow e^{-1} \mod (p' \cdot q')$
    $A \leftarrow Q^d \mod n$
    return $(A, e, v)$
```

corresponding to the public key $e$ is computed. Finally, $A = Q^d \mod n$ is the RSA signature over the aggregated messages. As a result the Camenisch-Lysyanskaya signature over the messages $\{m_i\}_{i \in \mathcal{M}}$ is the triple $(A, e, v)$ (see Algorithm 5.3).

In order to verify such a Camenisch-Lysyanskaya signature $(A, e, v)$ the RSA signature over the aggregated messages has to be verified. That is, the verifier has to check the following equation:

$$A^e = \frac{Z}{S^v \cdot \prod_{i \in \mathcal{M}} R_i^{m_i}} \mod n \tag{5.2}$$

This is equivalent to checking the following equation, as used in Algorithm 5.4, such that it is not necessary to compute the inverse:

$$Z = A^e \cdot S^v \cdot \textstyle\prod_{i \in \mathcal{M}} R_i^{m_i} \mod n \tag{5.3}$$

**Algorithm 5.4** Verify a basic Camenisch-Lysyanskaya signature.

```
function CL-VERIFY($\{m_i\}_{i \in \mathcal{M}}, (A, e, v), (n, S, Z, \{R_i\}_{i \in \mathcal{M}})$)
    $Z' \leftarrow A^e \cdot S^v \mod n$
    for each $i \in \mathcal{M}$ do
        $Z' \leftarrow Z' \cdot R_i^{m_i} \mod n$
    if $Z \neq Z'$ then
        return INVALID
    return VALID
```

### 5.1.3 Blind Signatures

Figure 5.1 depicts the protocol for generating blind Camenisch-Lysyanskaya signatures. This protocol hides the messages $\{m_i\}_{i \in \mathcal{M}_H}$, where $\mathcal{M}_H \subseteq \mathcal{M}$, from the signer by generating a commitment to these values and blinding them according to

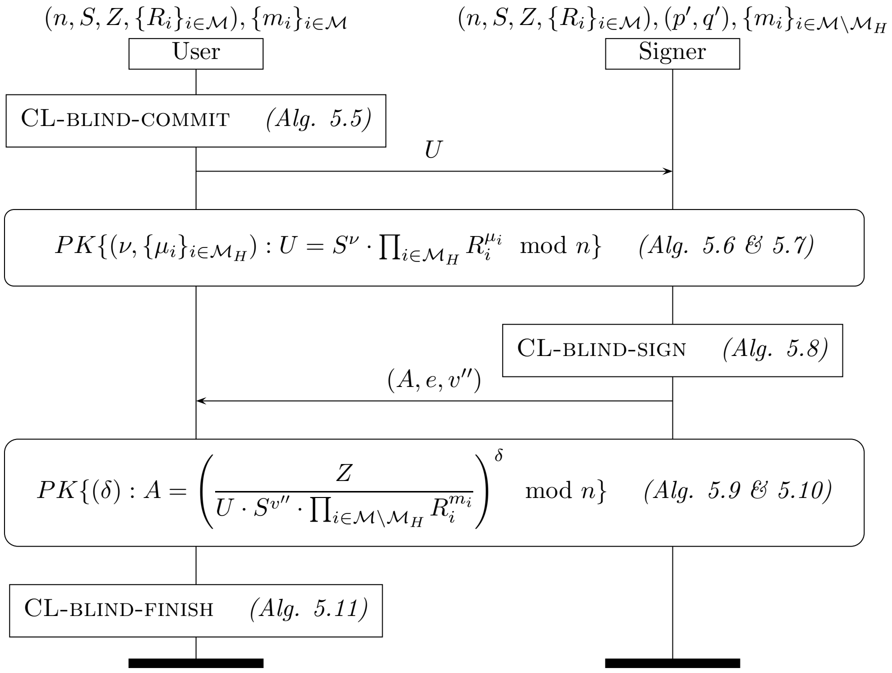

$(n, S, Z, \{R_i\}_{i \in \mathcal{M}}), \{m_i\}_{i \in \mathcal{M}}$
User
$(n, S, Z, \{R_i\}_{i \in \mathcal{M}}), (p', q'), \{m_i\}_{i \in \mathcal{M} \setminus \mathcal{M}_H}$
Signer
CL-BLIND-COMMIT *(Alg. 5.5)*
$U$
$PK\{(\nu, \{\mu_i\}_{i \in \mathcal{M}_H}) : U = S^\nu \cdot \prod_{i \in \mathcal{M}_H} R_i^{\mu_i} \mod n\}$ *(Alg. 5.6 & 5.7)*
CL-BLIND-SIGN *(Alg. 5.8)*
$(A, e, v'')$
$PK\{(\delta) : A = \left(\frac{Z}{U \cdot S^{v''} \cdot \prod_{i \in \mathcal{M} \setminus \mathcal{M}_H} R_i^{m_i}}\right)^\delta \mod n\}$ *(Alg. 5.9 & 5.10)*
CL-BLIND-FINISH *(Alg. 5.11)*

Figure 5.1: Protocol for generating blind signatures.

Algorithm 5.5. During this commitment phase these messages are aggregated into a single element $U$ and hidden by the blinding value $v'$. Note that the remaining messages $\{m_i\}_{i \in \mathcal{M} \setminus \mathcal{M}_H}$, that are not hidden during this phase, are known by the signer.

**Algorithm 5.5** Prepare for a blind Camenisch-Lysyanskaya signature.

```
1: function CL-BLIND-COMMIT($\{m_i\}_{i \in \mathcal{M}_H}, (n, S, Z, \{R_i\}_{i \in \mathcal{M}})$)
2:     $v' \leftarrow$ RANDOM( )
3:     $U \leftarrow S^{v'} \mod n$
4:     for each $i \in \mathcal{M}_H$ do
5:         $U \leftarrow U \cdot R_i^{m_i} \mod n$
6:     return $(U, v')$
```

In order to prove to the signer that the user actually knows the hidden messages, the following proof of knowledge has to be carried out.

$$PK\{(\nu, \{\mu_i\}_{i \in \mathcal{M}_H}) : U = S^\nu \cdot \textstyle\prod_{i \in \mathcal{M}_H} R_i^{\mu_i} \mod n\}$$

This proof not only proves that the user knows the hidden messages, but also that the value $U$ has been constructed correctly by the user. This can be implemented as an interactive zero-knowledge protocol or using Algorithms 5.6 and 5.7 which, respectively, construct and verify a non-interactive proof of knowledge. The freshness

**Algorithm 5.6** Generate a proof of correctness for $U$.

```
 1: function CL-PROVE-U($(U, v'), n_U, \{m_i\}_{i \in \mathcal{M}_H}, (n, S, Z, \{R_i\}_{i \in \mathcal{M}})$)
 2:     $\tilde{v}' \leftarrow$ Random( )
 3:     $\tilde{U} \leftarrow S^{\tilde{v}'} \mod n$
 4:     for each $i \in \mathcal{M}_H$ do
 5:         $\tilde{m}_i \leftarrow$ Random( )
 6:         $\tilde{U} \leftarrow \tilde{U} \cdot R_i^{\tilde{m}_i} \mod n$
 7:     $c \leftarrow$ Hash($U, \tilde{U}, n_U$)
 8:     $\hat{v}' \leftarrow \tilde{v}' + c \cdot v'$
 9:     for each $i \in \mathcal{M}_H$ do
10:         $\hat{m}_i \leftarrow \tilde{m}_i + c \cdot m_i$
11:     return $(c, \hat{v}', \hat{m}_{i \in \mathcal{M}_H})$
```

**Algorithm 5.7** Verify the proof of correctness for $U$.

```
 1: function CL-VERIFY-U($U, (c, \hat{v}', \{\hat{m}_i\}_{i \in \mathcal{M}_H}), n_U, (n, S, Z, \{R_i\}_{i \in \mathcal{M}})$)
 2:     $\hat{U} \leftarrow U^{-c} \cdot S^{\hat{v}'} \mod n$
 3:     for each $i \in \mathcal{M}_H$ do
 4:         $\hat{U} \leftarrow \hat{U} \cdot R_i^{\hat{m}_i} \mod n$
 5:     if $c \neq$ Hash($U, \hat{U}, n_U$) then
 6:         return Invalid
 7:     return Valid
```

of this proof is guaranteed by a nonce $n_U$ provided by the signer. The verification succeeds if the signer can successfully reconstruct the commitment $\tilde{U}$ on $U$. This reconstruction works because of the following equation.

$$\begin{aligned}
\hat{U} &= S^{\hat{v}'} \cdot U^{-c} \cdot \prod_{i \in \mathcal{M}_H} R_i^{\hat{m}_i} = S^{\tilde{v}'+c \cdot v'} \cdot U^{-c} \cdot \prod_{i \in \mathcal{M}_H} R_i^{\tilde{m}_i + c \cdot m_i} \\
&= S^{\tilde{v}'} \cdot S^{c \cdot v'} \cdot U^{-c} \cdot \prod_{i \in \mathcal{M}_H} R_i^{\tilde{m}_i} \cdot \prod_{i \in \mathcal{M}_H} R_i^{c \cdot m_i} \\
&= S^{\tilde{v}'} \cdot U^{-c} \cdot U^{c} \cdot \prod_{i \in \mathcal{M}_H} R_i^{\tilde{m}_i} = S^{\tilde{v}'} \cdot \prod_{i \in \mathcal{M}_H} R_i^{\tilde{m}_i} \\
&= \tilde{U} \mod n
\end{aligned}$$

The next step is the actual signing process. In this step the hidden messages, aggregated in $U$, are combined with the remaining known messages $\{m_i\}_{i \in \mathcal{M} \setminus \mathcal{M}_H}$ that will be included in the signature. This process is very similar to the basic signature operation as given in Algorithm 5.3. The main difference in Algorithm 5.8 is the inclusion of the value $U$ received from the user in the previous message aggregation step.

Furthermore, the signer provides the following proof of knowledge to show the

**Algorithm 5.8** Generate a blind Camenisch-Lysyanskaya signature.

```
1:  function CL-blind-sign($U, \{m_i\}_{i \in \mathcal{M} \setminus \mathcal{M}_H}, (n, S, Z, \{R_i\}_{i \in \mathcal{M}}), (p', q')$)
2:      $v'' \leftarrow$ Random( )
3:      $U \leftarrow U \cdot S^{v''} \mod n$
4:      for each $i \in \mathcal{M} \setminus \mathcal{M}_H$ do
5:          $U \leftarrow U \cdot R_i^{m_i} \mod n$
6:      $Q \leftarrow Z \cdot U^{-1} \mod n$
7:      $e \leftarrow$ RandomPrime( )
8:      $d \leftarrow e^{-1} \mod (p' \cdot q')$
9:      $A \leftarrow Q^d \mod n$
10:     return $(A, e, v'')$
```

user that the signature has been constructed correctly.

$$PK\{(\delta) : A = \left( \frac{Z}{U \cdot S^{v''} \cdot \prod_{i \in \mathcal{M} \setminus \mathcal{M}_H} R_i^{m_i}} \right)^{\delta} \mod n\}$$

Again, this proof of knowledge can be implemented as an interactive zero-knowledge protocol or as a non-interactive proof of knowledge. Algorithms 5.9 and 5.10, respectively, construct and verify such a non-interactive proof with a nonce $n_A$ provided by the user to guarantee the freshness. The commitment $\tilde{A}$ can be reconstructed to verify the proof according to the following equation[3].

$$\hat{A} = A^{c+\hat{d}\cdot e} = Q^{e^{-1}\cdot(c+\hat{d}\cdot e)} = Q^{c\cdot e^{-1}+\hat{d}} = Q^{c\cdot e^{-1}+\tilde{d}-c\cdot e^{-1}} = Q^{\tilde{d}} = \tilde{A} \mod n$$

**Algorithm 5.9** Generate a proof of correctness for $A$.

```
1:  function CL-prove-A($(A, e, v''), d, n_A, (n, S, Z, \{R_i\}_{i \in \mathcal{M}}), (p', q')$)
2:      $\tilde{d} \leftarrow$ Random( )
3:      $\tilde{A} \leftarrow Q^{\tilde{d}} \mod n$
4:      $c \leftarrow$ Hash($Q, A, \tilde{A}, n_A$)
5:      $\hat{d} \leftarrow \tilde{d} - c \cdot d \mod (p' \cdot q')$
6:      return $(c, \hat{d})$
```

Finally, the user has to complete the signature according to Algorithm 5.11 which combines the blinding values of the user and signer, $v'$ and $v''$ respectively, to become the randomisation value $v$ of the Camenisch-Lysyanskaya signature $(A, e, v)$. This signature can now be verified using the verification procedure from the basic signature scheme (Algorithm 5.4) or it can be used to prove knowledge of this signature as described in the following section.

[3]Note that in version 2.3.3 of the specification of the Identity Mixer cryptographic library [IBM11], the reconstructed value $\hat{A}$ is incorrect, after reporting this to the authors it has been corrected in version 2.3.4 [IBM12].

**Algorithm 5.10** Verify the proof of correctness for $A$.

```
function CL-VERIFY-A($(A, e, v''), (c, \hat{d}), n_A, (n, S, Z, \{R_i\}_{i \in \mathcal{M}})$)
    $Q \leftarrow A^e \mod n$
    $\hat{A} \leftarrow A^{c + \hat{d} \cdot e} \mod n$
    if $c \neq$ HASH($Q, A, \hat{A}, n_A$) then
        return INVALID
    return VALID
```

**Algorithm 5.11** Finish a blind Camenisch-Lysyanskaya signature.

```
function CL-BLIND-FINISH($v', (A, e, v'')$)
    $v \leftarrow v' + v''$
    return $(A, e, v)$
```

## 5.1.4 Proving Knowledge of a Signature

Blind signatures hide (a number of) the messages from the signer during signature generation. The goal of proving knowledge of a signature is to hide (a number of) the messages from the verifier during signature verification as well as to hide the actual value of the signature to prevent traceability. This process, as depicted in Figure 5.2, allows the user to prove that she has a signature over one or more (possibly hidden) messages without revealing the actual signature to the verifier.

To hide the Camenisch-Lysyanskaya signature $(A, e, v)$ and prevent linkability based on the signature values $A$, $e$, and $v$ the signature is randomised, using Algorithm 5.12. First a randomisation value $r$ is generated to randomise the RSA signature value $A$. Next the value $v$ is adjusted such that the signature remains valid, that is, it still satisfies (5.2):

$$\begin{aligned} A'^e &= (A \cdot S^r)^e = A^e \cdot S^{e \cdot r} \\ &= \frac{S^{e \cdot r} \cdot Z}{S^v \cdot \prod_{i \in \mathcal{M}} R_i^{m_i}} = \frac{S^{-e \cdot r} \cdot S^{e \cdot r} \cdot Z}{S^{-e \cdot r} \cdot S^v \cdot \prod_{i \in \mathcal{M}} R_i^{m_i}} \\ &= \frac{Z}{S^{v - e \cdot r} \cdot \prod_{i \in \mathcal{M}} R_i^{m_i}} = \frac{Z}{S^{v'} \cdot \prod_{i \in \mathcal{M}} R_i^{m_i}} \mod n \end{aligned}$$

**Algorithm 5.12** Randomise a Camenisch-Lysyanskaya signature.

```
function CL-RANDOMISE($(A, e, v), (n, S, Z, \{R_i\}_{i \in \mathcal{M}})$)
    $r \leftarrow$ RANDOM( )
    $A' \leftarrow A \cdot S^r \mod n$
    $v' \leftarrow v - e \cdot r$
    return $(A', e, v')$
```

This randomisation operation only effectively randomises the $A$ value of the signature. Hence it is required to hide the $e$ and $v'$ values using a zero-knowledge proof

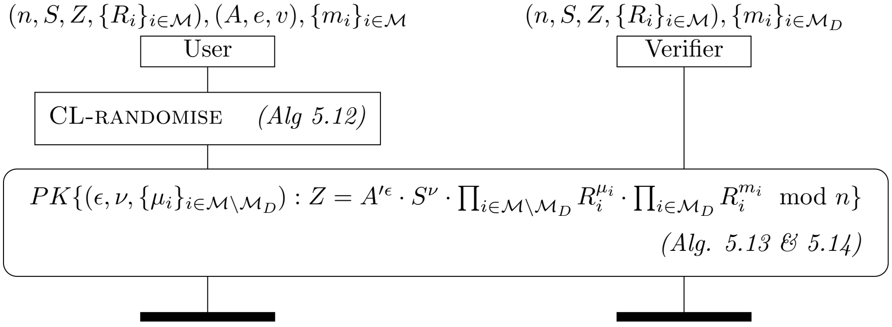

$(n, S, Z, \{R_i\}_{i\in\mathcal{M}}), (A, e, v), \{m_i\}_{i\in\mathcal{M}}$
User
$(n, S, Z, \{R_i\}_{i\in\mathcal{M}}), \{m_i\}_{i\in\mathcal{M}_D}$
Verifier
CL-RANDOMISE (Alg 5.12)
$PK\{(\epsilon, \nu, \{\mu_i\}_{i\in\mathcal{M}\setminus\mathcal{M}_D}) : Z = A'^{\epsilon} \cdot S^{\nu} \cdot \prod_{i\in\mathcal{M}\setminus\mathcal{M}_D} R_i^{\mu_i} \cdot \prod_{i\in\mathcal{M}_D} R_i^{m_i} \mod n\}$
(Alg. 5.13 & 5.14)

Figure 5.2: Protocol for proving knowledge of a signature.

when revealing this randomised signature and the messages $\{m_i\}_{i\in\mathcal{M}_D}$ disclosed to the verifier. Furthermore the following proof hides the messages $\{m_i\}_{i\in\mathcal{M}\setminus\mathcal{M}_D}$ that the user does not wish to disclose to the verifier.

$$PK\{(\epsilon, \nu, \{\mu_i\}_{i\in\mathcal{M}\setminus\mathcal{M}_D}) : Z = A'^{\epsilon} \cdot S^{\nu} \cdot \prod_{i\in\mathcal{M}\setminus\mathcal{M}_D} R_i^{\mu_i} \cdot \prod_{i\in\mathcal{M}_D} R_i^{m_i} \mod n\}$$

Algorithm 5.13 describes the operations that have to be performed to generate a proof of knowledge of a signature and the hidden messages. This proof can then be verified to check the correctness of the signature over the disclosed messages using Algorithm 5.14. Note that the messages $\{m_i\}_{i\in\mathcal{M}_D}$ disclosed by the user are known by the verifier and are input to the verification algorithm. Similar to the previous proofs in this scheme the verification relies on the reconstruction of the commitments, which in this case is possible because of the following equation.

**Algorithm 5.13** Prove knowledge of a Camenisch-Lysyanskaya signature.

```
1: function CL-PROVE-D($\{m_i\}_{i\in\mathcal{M}\setminus\mathcal{M}_D}, (A', e, v'), n_D, (n, S, Z, \{R_i\}_{i\in\mathcal{M}})$)
2:     $\tilde{e} \leftarrow$ RANDOM( )
3:     $\tilde{v} \leftarrow$ RANDOM( )
4:     $\tilde{Z} \leftarrow A'^{\tilde{e}} \cdot S^{\tilde{v}} \mod n$
5:     for each $i \in \mathcal{M} \setminus \mathcal{M}_D$ do
6:         $\tilde{m}_i \leftarrow$ RANDOM( )
7:         $\tilde{Z} \leftarrow \tilde{Z} \cdot R_i^{\tilde{m}_i} \mod n$
8:     $c \leftarrow$ HASH($A', \tilde{Z}, n_D$)
9:     $\hat{e} \leftarrow \tilde{e} + c \cdot e$
10:    $\hat{v} \leftarrow \tilde{v} + c \cdot v'$
11:    for each $i \in \mathcal{M} \setminus \mathcal{M}_D$ do
12:        $\hat{m}_i \leftarrow \tilde{m}_i + c \cdot m_i$
13:    return $(c, A', \hat{e}, \hat{v}, \{\hat{m}_i\}_{i\in\mathcal{M}\setminus\mathcal{M}_D})$
```

**Algorithm 5.14** Verify the signature proof of knowledge.

```
1: function CL-verify-D($(c, A', \hat{e}, \hat{v}, \{\hat{m}_i\}_{i \in \mathcal{M} \setminus \mathcal{M}_D}, \{m_i\}_{i \in \mathcal{M}_D}), n_D, (n, S, Z, \{R_i\}_{i \in \mathcal{M}})$)
2:     $\hat{Z} \leftarrow Z^{-c} \cdot A'^{\hat{e}} \cdot S^{\hat{v}} \mod n$
3:     for each $i \in \mathcal{M}_D$ do
4:         $\hat{Z} \leftarrow \hat{Z} \cdot R_i^{c \cdot m_i} \mod n$
5:     for each $i \in \mathcal{M} \setminus \mathcal{M}_D$ do
6:         $\hat{Z} \leftarrow \hat{Z} \cdot R_i^{\hat{m}_i} \mod n$
7:     if $c \neq$ Hash($A', \hat{Z}, n_D$) then
8:         return Invalid
9:     return Valid
```

$$\begin{aligned}
\hat{Z} &= Z^{-c} \cdot A'^{\hat{e}} \cdot S^{\hat{v}} \cdot (\textstyle\prod_{i \in \mathcal{M}_D} R_i^{m_i})^c \cdot \prod_{i \in \mathcal{M} \setminus \mathcal{M}_D} R_i^{\hat{m}_i} \\
&= Z^{-c} \cdot A'^{\tilde{e}+c \cdot e} \cdot S^{\tilde{v}+c \cdot v'} \cdot \textstyle\prod_{i \in \mathcal{M}_D} R_i^{c \cdot m_i} \cdot \prod_{i \in \mathcal{M} \setminus \mathcal{M}_D} R_i^{\tilde{m}_i + c \cdot m_i} \\
&= Z^{-c} \cdot A'^{\tilde{e}} \cdot A'^{c \cdot e} \cdot S^{\tilde{v}'} \cdot S^{c \cdot v'} \cdot \textstyle\prod_{i \in \mathcal{M}_D} R_i^{c \cdot m_i} \cdot \prod_{i \in \mathcal{M} \setminus \mathcal{M}_D} R_i^{\tilde{m}_i} \cdot \prod_{i \in \mathcal{M} \setminus \mathcal{M}_D} R_i^{c \cdot m_i} \\
&= Z^{-c} \cdot (A'^{e} \cdot S^{v'} \cdot \textstyle\prod_{i \in \mathcal{M}} R_i^{m_i})^c \cdot A'^{\tilde{e}} \cdot S^{\tilde{v}'} \cdot \prod_{i \in \mathcal{M} \setminus \mathcal{M}_D} R_i^{\tilde{m}_i} \\
&= (A'^{e} \cdot S^{v'} \cdot \textstyle\prod_{i \in \mathcal{M}} R_i^{m_i})^{-c} \cdot (A'^{e} \cdot S^{v'} \cdot \prod_{i \in \mathcal{M}} R_i^{m_i})^c \cdot A'^{\tilde{e}} \cdot S^{\tilde{v}'} \cdot \prod_{i \in \mathcal{M} \setminus \mathcal{M}_D} R_i^{\tilde{m}_i} \\
&= A'^{\tilde{e}} \cdot S^{\tilde{v}'} \cdot \textstyle\prod_{i \in \mathcal{M} \setminus \mathcal{M}_D} R_i^{\tilde{m}_i} \\
&= \tilde{Z} \mod n
\end{aligned}$$

Note that this equation uses (5.3) and hence depends on the validity of the signature used to generate this proof. If the triple $(A, e, v)$ is not a valid signature, that is (5.3) does not hold, the proof will fail.

## 5.2 Identity Mixer Credentials

The Identity Mixer technology is tightly built upon the Camenisch-Lysyanskaya signature scheme and its protocols. The blind signature scheme provides the issuance protocol where the messages become the contents of the credential. The protocol for proving knowledge of a signature is used as the credential verification protocol.

An Identity Mixer credential contains a master secret $s$, which belongs to the user and is never revealed, and a collection of attributes $\{a_i\}_{i \in \mathcal{A}}$, where $\mathcal{A}$ denotes the set of attribute indices. Hence, the sets of message indices for the issuance protocol become $\{m_i\}_{i \in \mathcal{M}_H} = \{s\}$ and $\mathcal{M} = \mathcal{M}_H \cup \mathcal{A}$. As a result of the issuance protocol the user obtains a signature $(A, e, v)$ over these values which completes the credential.

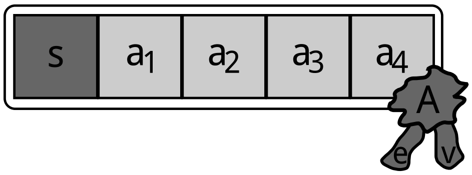


Figure 5.3: A visual representation of an Identity Mixer credential.

To summarise, an Identity Mixer credential, as depicted in Figure 5.3, consists of:

- the user's master secret $s$,
- a collection of attributes $\{a_i\}_{i \in \mathcal{A}}$, and
- a signature $(A, e, v)$, over the user's master secret and the attributes, where $A = \left( \frac{Z}{S^v \cdot R_s^s \cdot \prod_{i \in \mathcal{A}} R_i^{a_i}} \right)^d \mod n$ and $d = e^{-1} \mod (p' \cdot q')$.

This credential can now be used with the protocol for proving knowledge of a signature to selectively disclose the attributes to a verifier. In this case the set of disclosed messages $\mathcal{M}_D \subseteq \mathcal{A}$ is a selection of the attributes contained in the credential. Using Algorithm 5.13 the user can now generate a proof of knowledge that can be sent to the verifier in order to reveal the attributes and to prove that they are signed by the issuer.

This is the core of the Identity Mixer system that we have implemented on a smart card, for extensions of this proof which provide more features we refer the reader to the specification of the Identity Mixer technology [IBM12].

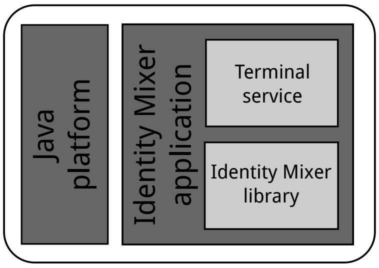
### 5.2.1 Direct Anonymous Attestation

Direct anonymous attestation [BCC04a] is a technology based on Identity Mixer, but it omits the attributes, hence $\mathcal{M} = \mathcal{M}_H$ and $\mathcal{M}_D = \emptyset$. It allows a user to convince a verifier that she uses a platform that has embedded a certified hardware module[4] by proving knowledge of the signature that certifies the module. The protocol protects the user's privacy: if she talks to the same verifier twice, the verifier is not able to tell whether or not he communicates with the same user as before or with a different one.

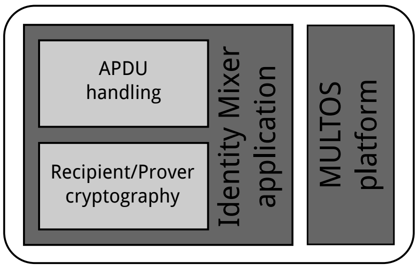
## 5.3 Identity Mixer on Smart Cards

We are not the first to develop an implementation of the Identity Mixer on a smart card. In 2009 Bichsel etal. [BCGS09] implemented a minimal Identity Mixer system on a Java Card whereas Sterckx etal. [SGPV09] did the same for direct anonymous attestation. They provide the first implementations of this technology on smart cards. The major drawback of these implementations is the running time of several seconds which is still too much for being really practical.

[4]Direct anonymous attestation has been adopted in 2004 by the Trusted Computing Group in the Trusted Platform Module specification as the method for remote authentication of a hardware module.



(a) Terminal



(b) Card

Figure 5.4: System architecture for Identity Mixer on a smart card.

Our system consists of two parts, depicted in Figure 5.4, a terminal (a) which interacts with a card (b) using APDUs. The terminal application is written in Java and uses the Identity Mixer cryptographic library[5] provided by IBM Research. Thanks to Patrick Bichsel, version 2.3.4 of this library provides interfaces for the various roles in the protocols such that we could create an extension to this library, the terminal service[6], which takes care of all smart card specifics. The service implements the user roles of the Identity Mixer issuance and verification protocols, described by the `Recipient` and `Prover` interfaces respectively. These interfaces are implemented by translating all Java method calls from the library into the corresponding APDU commands and converting the Java data types to raw byte arrays, suitable for APDU communication.

The Identity Mixer application[7] on the card takes care of handling the incoming APDUs and storing the values into the internal data structures. While handling the communication with the terminal is the largest part of the application, the main part is the implementation of the cryptographic operations for the Identity Mixer protocols. This allows the card to perform the user roles without depending on the terminal for any computations or proof generation. The only thing the terminal is responsible for is providing the data in the correct format.

## 5.3.1 Smart Card Implementation

Just as with the U-Prove implementation, we have chosen to use the MULTOS C interface to do our prototype implementation of Identity Mixer. The programming environment is convenient for smart card programming and allows us some more flexibility for memory management. It will be explained below why this is crucial.

Implementing the Identity Mixer specification did not turn out to be that hard in the beginning, the available API makes it easy to implement the cryptographic protocol. In principle, it is a direct translation from the mathematical description to API calls. The only API restriction we came across was that the `ModularExponentiation` function does not accept exponents larger than the modulus size. In our case

[5]The library is available for download at `https://prime.inf.tu-dresden.de/idemix/` while our patches can be found at `https://github.com/credentials/idemix_library`.

[6]`https://github.com/credentials/idemix_terminal`

[7]`https://github.com/pimvullers/idemix_multos/`

this only involves exponentiations with base $S$ (see details in Section 5.1). Hence we added a function `SpecialModularExponentiation` which implements the same method as used by Bichsel etal. [BCGS09], that is, splitting one exponentiation up into two[8] exponentiations and one multiplication.

Our initial implementation only used static memory to store the variables. This allowed us to work without thinking about which memory segment to use. The drawback of this approach was a bad performance caused by the EEPROM memory which takes a long time, compared to RAM, to write new values.

Once we had a functionally correct implementation, we started to optimise. This was done by moving the buffer, which stores the intermediate results for larger computations, to the public memory. The next step was to move the session variables to the dynamic memory, which required careful organisation due to the limited amount of available storage. However, when the dynamic memory use increased the stack-based execution model started to cause trouble.

Initially, the stack could use the full size of the dynamic memory, such that we had sufficient space to use functions and put the (relatively) large input values on the stack. When using the C-interface, the compiler takes care of managing the stack and putting input values on it when an instruction needs this. However, this makes it difficult to get an idea on how much space the stack actually needs. By trial and error, we discovered that we used quite some amount of memory for the stack, which left us with only limited amount of space for session variables.

To improve this situation, we reduced the number of function calls by inlining some convenience functions. We also switched to using global variables instead of function parameters, such that when we use a function, it does not require much space on the stack. To get the last few, often used, variables into RAM, we decided to split up some computations into smaller parts such that the values to be put on the stack also get smaller. For example, additions can be computed using addition with carry, and multiplications using grade-school multiplication. This adds a number of extra operations, but it is worth the memory gain which results in improved performance.

Finally, we translated most parts of the cryptographic C code to MULTOS assembly code which allowed us to optimise the use of stack and reduce the amount of memory operations during calculations. This was required since the provided C-functions moved the values from the stack to the variable locations, while they are needed again in the next operation. By doing this we could reduce the amount of memory required for intermediate values which in the end allowed us to getall necessary variables in RAM.

## 5.4 Performance Results

There are two important performance measures: the time it takes to issue a new credential to the card, and the running time of the verification protocol.

For these performance tests we've used, where possible, the same test vectors as with the U-Prove implementation in order to get comparable results (see Chapter 6

[8]This method actually requires three exponentiations, but we can precompute one exponentiation during initialisation, since we only need this method for the base $S$.

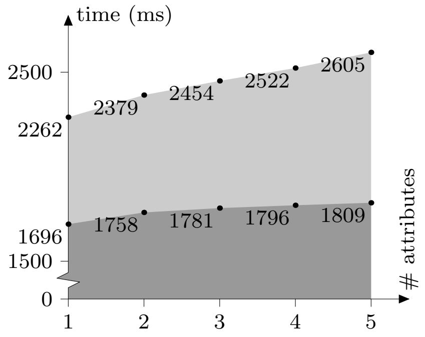


Figure 5.5: Identity Mixer credential issuance times for various numbers of attributes (▩: computation time, ▨: overhead).

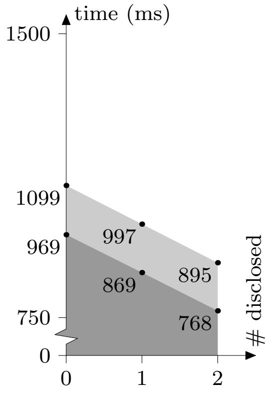
for a detailed comparison). Furthermore, our implementation uses a modulus size of 1024 bits, which provides a minimal level of security, but an acceptable performance. Given the amount of RAM on the card and this size of the modulus, and hence the size of all group elements, we can support at most 5 attributes per credential in our implementation.

For issuance, we measured the running time of the protocol and determined how much time was actually spent on computations and which part was used to transfer and store the values (marked as overhead; see Figure 5.5). From these results we can conclude that the number of attributes included in a credential has only a minimal effect on the computations. An increase in the number of attributes does, however, result in an increase of the overhead of approximately 100 milliseconds per attribute. This is caused by the additional data that has to be transferred and stored on the card.

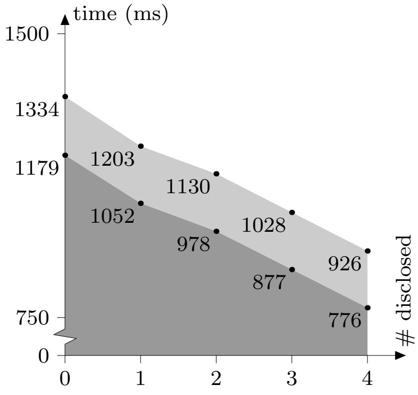
Sterckx etal. [SGPV09] implement the direct anonymous attestation protocol, which is derived from the Identity Mixer protocols, on a Java Card. With a 1024 bits modulus they achieve a running time of 2.4 seconds of which 19% is overhead, which gives a computation time of approximately 1.9 seconds. This is good, and in line with the results we got, but unfortunately the direct anonymous attestation protocol does not support any attributes as it is just targeted at anonymous authentication and hence only uses a secret key.

Bichsel etal. [BCGS09] from IBM Research Zürich also implemented a variant of the direct anonymous attestation protocol on a Java Card. They report a running time of 7.4 seconds for a modulus size of 1280 bits, which is larger than the 1024 bits we used. It is unclear, however, which transaction time they measured. But again, this implementation does not include any attributes.

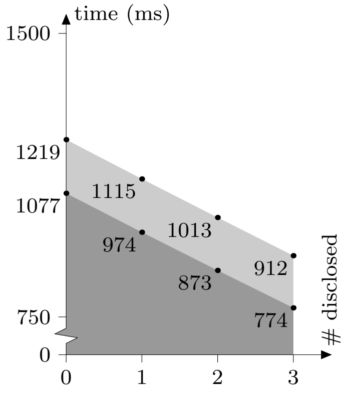
For selective disclosure, we measured the running times of four configurations (see Figure 5.6). These configurations have been chosen because two is the smallest number of attributes for which selective disclosure makes sense and five is the largest number of attributes that we can currently keep in memory while not using EEPROM for computations during the transaction. By comparing these graphs, it is clear that each attribute that is disclosed reduces the computation time with roughly 100 milliseconds. Also, the scenario in which all attributes are disclosed



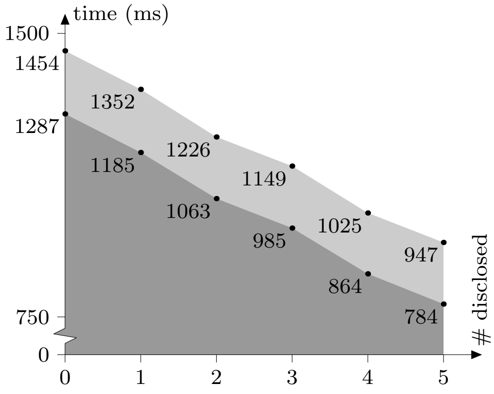
(a) 2 stored attributes



(b) 4 stored attributes



(c) 3 stored attributes



(d) 5 stored attributes

Figure 5.6: Identity Mixer credential verification times for different configurations (▓: computation time, ░: overhead).

results in similar computation times, with only slightly increased overhead for the large number of attributes that have to be sent to the terminal.

Comparing our work with the implementations by Sterckx et al. [SGPV09] and Bichsel et al. [BCGS09] makes no real sense since they do not offer selective disclosure of attributes. It is, however, clear that our implementation provides a significant performance improvement over these implementations. For example, Sterckx etal. need 4.2 seconds to hide only the secret key while our implementation can hide the secret key and two attributes of a single credential in 1.1 seconds.

Finally, we take a look at the memory requirements for our implementation. To store a credential with two attributes 1.4KB of EEPROM is required. Each additional attribute for this credential requires another 160 bytes. This means that a card can contain approximately 30 credentials (when we store five attributes per credential). The large size of the credential values also has an impact on the amount of memory required to perform the computations. For the verification protocol this amounts to 1.9KB of memory and 1.5KB for the issuance protocol. We ended up using the APDU buffer as scratch area in order to perform all calculations in RAM. However, recent research [dlPHV14] shows that these memory requirements can be reduced at the expense of an increased running time. This allows the implementation to be extended with additional functionality whereas this is not possible in the current implementation described in this chapter.

# Chapter 6

# Discussion

In the previous chapters we described the attribute-based credential technologies and the performance of their smart card implementations individually. In this chapter we compare them with each other to determine their (relative) strong and weak points.

## 6.1 Attribute-based Credential Technologies

When we compare the technologies on the cryptographic level it is clear that different approaches can be taken to achieve the same goal. During issuance both U-Prove and Identity Mixer use a blind signature protocol to construct a credential, whereas the self-blindable credentials use a regular signature scheme. This can be explained by the fact that in the latter case the user's public key is used for issuing the credential, whereas the former involves the user's secret. Another difference at this stage is that Identity Mixer supports committed values to be included in the credential while U-Prove and the self-blindable credentials require all the attribute values to be known to the issuer[1].

Issuer unlinkability is satisfied by each of the technologies. Identity Mixer and the self-blindable credentials require the issuer's signature in the credential to be randomised to achieve this, while the U-Prove issuance protocol results in an unlinkable signature. The signature randomisation also provides (native) multi-show unlinkability, whereas the U-Prove technology requires multiple signatures in order to provide this privacy property.

When it comes to credential verification, the self-blindable credentials provide the most basic scheme. Selective disclosure of attributes is not applicable since a credential only contains a single attribute statement. Also, because the attribute statement is contained in a regular certificate it cannot be used to prove properties of the attribute value, such as proving that the current date is within the validity period specified in the attribute. Both U-Prove and Identity Mixer offer selective disclosure of attributes and support the construction of zero-knowledge proofs in which properties of the attributes can be proved.

[1]The U-Prove technology does provide an information field that is hidden from the issuer, but this field is always disclosed during verification which makes it different form a regular attribute.

Table 6.1: Comparison of attribute-based credential technologies.

| | Self-blindable Credentials | U-Prove | Identity Mixer |
|---|---|---|---|
| issuer unlinkability | ✓ | ✓ | ✓ |
| multi-show unlinkability | ✓ | indirectly[2] | ✓ |
| blind signatures | | ✓ | ✓ |
| committed values | | | ✓ |
| signature randomisation | ✓ | | ✓ |
| selective disclosure | | ✓ | ✓ |
| zero-knowledge proofs | | ✓ | ✓ |
| elliptic curve cryptography | ✓ | not used | not supported[3] |

Another aspect is the type of cryptography used by the different technologies. Because the self-blindable credentials are based on elliptic curve cryptography, they are rather compact and require less communication between smart card and terminal. U-Prove can also be implemented using elliptic curve cryptography, since it relies on the discrete logarithm problem. While this would have been an ideal solution to eliminate the communication overhead, we do not have any cards that provide a proper interface which would allow us to implement U-Prove based on elliptic curve cryptography. Still, the prime-order subgroup construction used by U-Prove provides an advantage over the RSA-based approach used by Identity Mixer: it allows for modular reduction of the exponents in the protocols, which results in smaller session variables.

The above mentioned differences and similarities are summarised in Table 6.1. Note that this overview only focuses on the aspects of the technologies that we looked into. Other interesting overviews are provided by Lapon et al. [LKdDN11, Lap12], focusing on revocation strategies, and Corella [Cor11a, Cor11b], focusing on the use of attribute-based credentials in the context of the United States national strategy for trusted identities in cyberspace.

## 6.2 Attribute-based Credentials on Smart Cards

While we are not the first to implement attribute-based credentials on smart cards, we do provide, to the best of our knowledge, the most efficient implementations (see Section 6.3) with the most functionality. When we consider the existing implementations we can distinguish a few different approaches concerning the use of smart cards.

[2]Multi-show unlinkability for U-Prove can be realised by issuing multiple tokens for the same set of attributes which can later be verified independently.

[3]Camenisch and Lysyanskaya [CL04] also describe an elliptic curve based signature scheme which can serve as a basis for attribute-based credentials, but this is not used in Identity Mixer.

Table 6.2: Comparison between the *device-protection of credentials* approach and the *credentials on a smart card* approach.

| | device-protection of credentials | credentials on a smart card |
|---|---|---|
| characteristics | add-on security measure | full protocol implementation |
| card stores | only the device-protection attribute or secret | all attributes, other credential values |
| card computes | short zero-knowledge proof for the device-protection attribute | complete issuance and verification protocols |
| advantages | fast, lightweight, protect any number credentials using a single card pre-issued devices | independent use of the card, no need to trust the terminal |
| disadvantages | trusted terminal required | requires more card resources |

First, a smart card can be used as a means of *hardware-protection* for a credential. In this scenario the card performs only *a fraction* of the issuance and verification protocols. This is motivated by the constrained resources of smart cards. Brands [Bra00, Chapter 6] proposes to use this method for smart card integration in the U-Prove technology [Paq11c] (see Section 4.5), whereas Bichsel [Bic07] uses a similar construction to implement protection for Identity Mixer credentials.

In strong contrast to the minimalist hardware-protection implementation, Tews and Jacobs [TJ09] developed an implementation of attribute-based credentials that performs *all* operations on a smart card. A comparison between these approaches is given in Table 6.2. This *credentials on a smart card* approach does require more resources on the card, but it also solves the main disadvantage of the hardware-protection approach: the smart card cannot be used independently, since it is tied to computational (and storage) resources external to the card. This means that it requires a specific, card matching terminal, like the user's PC, to run the protocols.

The remaining implementations are based on *anonymous authentication* of the smart card after which it is trusted to provide valid attribute statements. To this end, Balasch [Bal08] and Sterckx [SGPV09] have implemented direct anonymous attestation (see Section 5.2.1) and Bichsel et al. [BCGS09] have chosen to implement Identity Mixer without any attributes. These solutions use the unlinkability properties of the Camenisch-Lysyanskaya signature scheme to achieve the anonymous authentication of the card. This anonymous authentication approach is also used in the German identity card (nPA) [BKMN10], except that a different authentication scheme is used (see Section 1.2.4).

## 6.3 Smart Card Performance

When comparing the smart card implementations we have to take into account that they have been developed on different platforms, and more importantly on different hardware. To be precise, the self-blindable credentials applet runs on an NXP SmartMX J3A081 chip with the Java Card platform (JCOP v2.4.1 R3), the

Table 6.3: Performance comparison between the NXP SmartMX chip and the Infineon SLE66 and SLE78 chips (time in milliseconds for 100 successive operations).

| | SmartMX | | SLE66 | | SLE78 | |
|---|---|---|---|---|---|---|
| | contact | wireless | contact | wireless | contact | wireless |
| SHA-1 RAM | 1110 | 1136 | 5120 | 5274 | 866 | 943 |
| SHA-1 EEPROM | 1442 | 1466 | 6125 | 6308 | 1188 | 1285 |
| RSA-1024 RAM | 772 | 777 | 1016 | 1060 | 668 | 877 |
| RSA-1024 EEPROM | 1941 | 1952 | 2936 | 3041 | 2449 | 2898 |
| RSA-2048 RAM | 1926 | 1950 | 14289 | 14898 | 1055 | 1449 |
| RSA-2048 EEPROM | 3838 | 3865 | 17237 | 17956 | 3473 | 4192 |

U-Prove application runs on an Infineon SLE66 chip with the MULTOS platform (I4F(1-1-2) on 360PE(M)) and the Identity Mixer application runs on the Infineon SLE78 chip with the MULTOS platform (ML3-36K-R1).

In order to compare the raw performance of the underlying hardware we developed a small test application that performs some basic operations[4] and stores the outcome either in RAM or EEPROM. The results of these tests are summarised in Table 6.3. From this we can conclude that the SLE66 is overall slower than the other cards, with a huge performance penalty when computing an RSA operation (which is basically a modular exponentiation) with a modulus of 2048 bits[5]. The timings of the SLE78 and SmartMX chips are much closer to each other. Here we notice that the SLE78 is in principle faster than the SmartMX, but also that there is a larger difference between the contact and wireless (or contactless) interfaces as well as between the memory used (RAM or EEPROM).

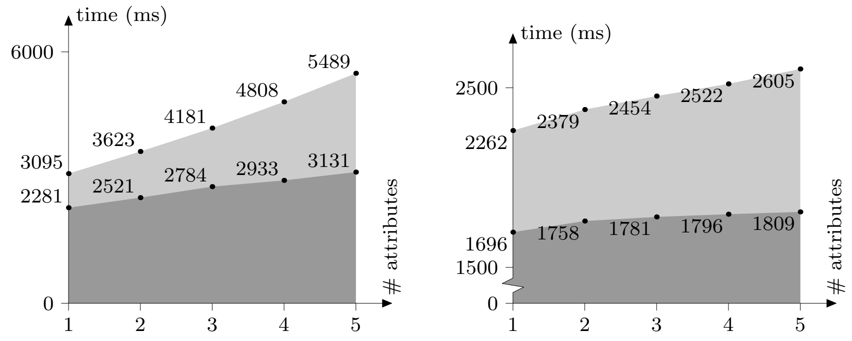
## 6.3.1 Credential Issuance

With this in mind we first look at the performance of the implementations during issuance. The issuance process for the self-blindable credentials does not involve any computations on the card, since it only has to store the credential. This makes it the most efficient implementation at this point since both U-Prove and Identity Mixer have to perform a blind signature protocol to issue credentials. To make the comparison between those two easier we put their issuance performance graphs (Figure 4.5 and Figure 5.5) next to each other in Figure 6.1.

Based on these graphs it is clear that the Identity Mixer implementation offers a clear improvement over the U-Prove issuance times. Not only are the absolute values better, the extra time it takes to issue more attributes does not increase as much as with the U-Prove implementation. However, we also need to take the hardware platform (Infineon SLE66 vs SLE78) into account, which in this case is in favour of the Identity Mixer implementation. Furthermore, our U-Prove implementation is

[4]A message digest computation using SHA-1; and two RSA encryptions using a random public key (with random exponents), one with a 1024 bits modulus and one with a 2048 bits modulus.

[5]Probably the cryptographic co-processor of the SLE66 chip does not support this operation directly such that a software solution is needed to handle operations involving a 2048 bits modulus.



(a) U-Prove (b) Identity Mixer

Figure 6.1: Credential issuance performance for various numbers of attributes (■: computation time, □: overhead).

built according to the first revision [Paq11a] of version 1.1 of the U-Prove cryptographic specification, whereas later revisions [PZ13], include an optimised issuance protocol in which the computation of the signature value $z'$ is mostly performed by the issuer (as described in Section 4.3). Taking this optimisation and a switch to the SLE78 platform into account, the U-Prove implementation should get a reasonable improvement in its issuance performance bringing it closer to the results we got from the Identity Mixer implementation.

### 6.3.2 Selective Disclosure of Attributes

Since a self-blindable credential only contains a single attribute, there is no selective disclosure option in the credential verification protocol. In this case, the user just chooses which attribute(s) to reveal, and hence which credential(s) to use. Thus, to show multiple attributes the protocol has to be performed multiple times, which also means that the transaction time is multiplied. This in strong contrast to the other technologies, where revealing more attributes, from the same credential, actually reduces the running time, as can be seen in Figure 6.2.

Comparing the results from Figures 6.2 and 6.3, it is clear that the U-Prove technology offers the best verification performance. The computation time for the verification of a U-Prove credential is overall lower than the computation times of the other technologies, even when a U-Prove credential contains five attributes of which only a single attribute is revealed. This performance could even be improved when the SLE78 chip can be used instead of the SLE66. Hence we can conclude that the signature randomisation performed by Identity Mixer and the self-blindable credentials has a significant impact on the running time of the verification protocol.

While the multi-show unlinkability property has a negative effect on the running time for Identity Mixer and the self-blindable credentials, it requires less storage on the card then U-Prove. This is due to the fact that to achieve multi-show unlinkability the card has to store multiple U-Prove tokens. Given that storage space is rather limited on smart cards[6], this is a serious drawback of the U-Prove technology.

[6]A typical modern smart card only has 36 to 144 KB of EEPROM for storing application data.

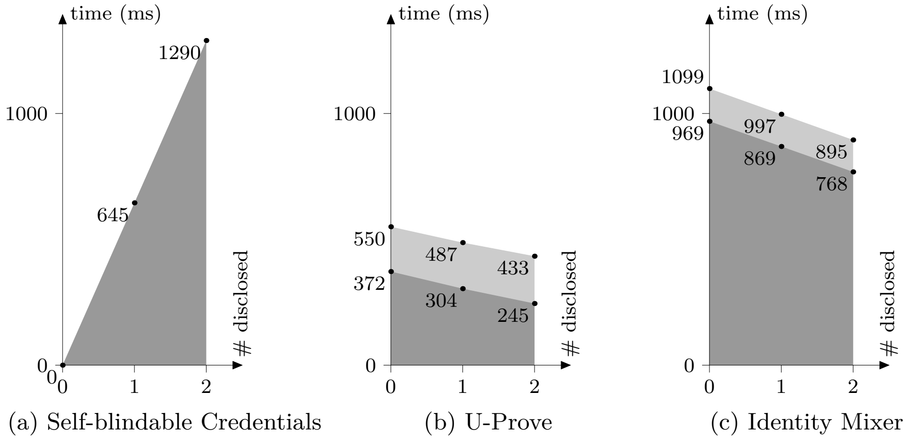


Figure 6.2: Credential verification performance with two attributes (■: computation time, ▨: overhead).

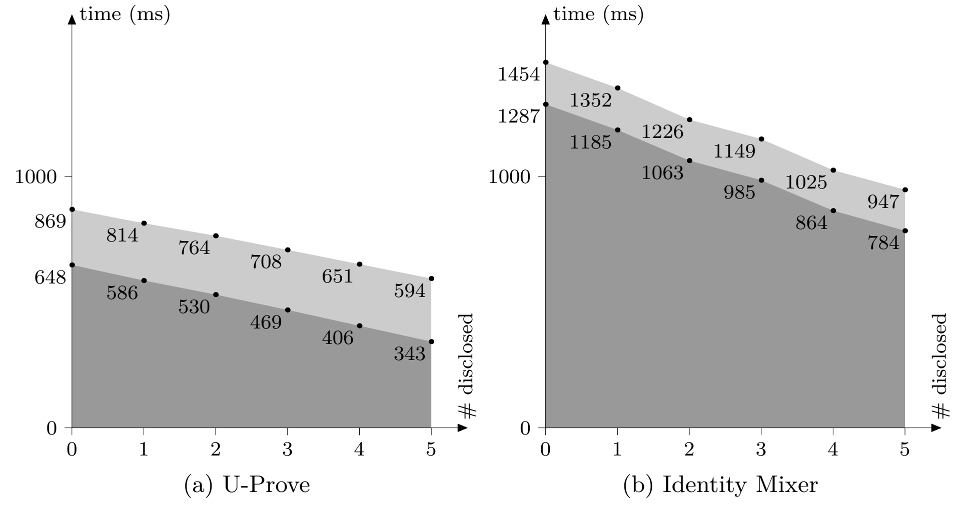


Figure 6.3: Credential verification performance with five attributes (■: computation time, ▨: overhead).

## 6.4 Final Remarks

The goal of the research presented in this thesis has been to

*develop efficient smart card implementations of attribute-based credentials*

and

*compare various cryptographic systems for attribute-based credentials.*

Compared to the existing smart card implementations mentioned in the previous section, we have made a clear performance improvement with all of our implementations. These implementations are not only faster, but also provide *full* credentials on a smart card instead of the partial solutions that have been developed to cope with the smart card shortcomings. With transaction times around, or even below one second we can conclude that these are, to the best of our knowledge, the first implementations that offer an acceptable performance for practical use.

The development of these prototypes allowed us to analyse the different technologies both from a technical and functional perspective. Due to the maturity and the multi-show unlinkability feature of the Identity Mixer technology that smart card implementation has been selected for use in a pilot project called *I Reveal My Attributes* or IRMA for short. This pilot aims to gain more experience in the practical use of these kinds of privacy-preserving technologies and the usability of smart card implementations therein. Please visit `https://www.irmacard.org/` for more information and the latest news on this project.

This IRMA project is a direct consequence of the demonstrated efficiency of our smart card implementations, in particular for Identity Mixer. This shows the impact and innovative power of our work in privacy-friendly identity management. The IRMA project has the wider goal of demonstrating the broad applicability of attribute-based credential technologies, and of the availability of a viable alternative for current smart card-based solutions. The availability of these privacy-friendly alternatives and the growing interest for privacy-by-design, in particular in privacy regulations, should lead to further innovations that can replace the traditional privacy-unfriendly identity solutions (typically based on unique identifiers).

# Bibliography

[AJ13] Gergely Alpár and Bart Jacobs. Credential design in attribute-based identity management. In Ronald Leenes and Eleni Kosta, editors, *Bridging distances in technology and regulation*, pages 189–204. Wolf Legal Publishers, April 2013. (Cited on page 2)

[Bal08] Josep Balasch. Smart card implementation of anonymous credentials. Master's thesis, KU Leuven, Belgium, 2008. (Cited on page 69)

[BCC04a] E. F. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In B. Pfitzmann and P. Liu, editors, *CCS 2004*, pages 132–145. ACM, October 2004. (Cited on pages 6 and 61)

[BCC04b] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick Drew McDaniel, editors, *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004*, pages 132–145. ACM, October 2004. (Cited on pages 51 and 52)

[BCGS09] Patrik Bichsel, Jan Camenisch, Thomas Groß, and Victor Shoup. Anonymous credentials on a standard Java Card. In *CCS 2009*, pages 600–610. ACM, November 2009. (Cited on pages 6, 8, 61, 63, 64, 65, and 69)

[BF01] D. Boneh and M.K. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, 2001. (Cited on page 20)

[BHJ$^+$10] Lejla Batina, Jaap-Henk Hoepman, Bart Jacobs, Wojciech Mostowski, and Pim Vullers. Developing efficient blinded attribute certificates on smart cards via pairings. In Dieter Gollmann and Jean-Louis Lanet, editors, *Smart Card Research and Advanced Applications, 9th IFIP WG 8.8/11.2 International Conference, CARDIS 2010, Passau, Germany, April 13-16, 2010. Proceedings*, volume 6035 of *Lecture Notes in Computer Science (LNCS)*, pages 209–222. Springer-Verlag, April 2010. (Cited on pages 5, 10, 25, and 33)

[Bic07] P. Bichsel. Theft and misuse protection for anonymous credentials. Master's thesis, ETH Zürich, Switzerland, November 2007. (Cited on page 69)

[BKMN10] Jens Bender, Dennis Kügler, Marian Margraf, and Ingo Naumann. Privacy-friendly revocation management without unique chip identifiers for the German national ID card. *Computer Fraud & Security*, September 2010. (Cited on pages 6 and 69)

[BL12] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. *IACR Cryptology ePrint Archive*, 2012:298, 2012. (Cited on page 5)

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science (LNCS)*, pages 514–532. Springer Berlin Heidelberg, 2001. (Cited on pages 25 and 26)

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004. (Cited on pages 20, 25, and 26)

[BN06] P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography - SAC 2005*, volume 3897 of *LNCS*, pages 319–331. Springer, 2006. (Cited on page 20)

[BP10] Stefan Brands and Christian Paquin. U-Prove cryptographic specification v1.0. Technical report, Microsoft Corporation, March 2010. (Cited on page 37)

[Bra00] Stefan A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy.* MIT Press, United States, 2000. (Cited on pages 5, 37, 45, and 69)

[BSS05] I. Blake, G. Seroussi, and N. P. Smart, editors. *Advances in Elliptic Curve Cryptography.* Number 317 in LMS. Cambridge Univ. Press, 2005. (Cited on page 20)

[Bun10] Bundesamt für Sicherheit in der Informationstechnik. Advanced security mechanisms for machine readable travel documents, Version 2.05. Technical Report TR-03110, German Federal Office for Information Security (BSI), Bonn, Germany, 2010. (Cited on pages 6 and 8)

[Cam07] Jan Camenisch. Direct anonymous attestation explained. Technical report, IBM Research, July 2007. (Cited on pages 22, 51, and 53)

[CEvdG88] David Chaum, Jan-Hendrik Evertse, and Jeroen van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In David Chaum and Wyn L. Price, editors, *Advances in Cryptology – EUROCRYPT87*, volume 304 of *Lecture Notes in Computer Science*, pages 127–141. Springer Berlin Heidelberg, 1988. (Cited on page 37)

[CG05] Jan Camenisch and Jens Groth. Group signatures: Better efficiency and new theoretical aspects. In Carlo Blundo and Stelvio Cimato, editors, *Security in Communication Networks, 4th International Conference, SCN 2004, Amalfi, Italy, September 8-10, 2004*, volume 3352 of *Lecture Notes in Computer Science*, pages 120–133. Springer, 2005. (Cited on pages 51 and 52)

[Cha83] David Chaum. Blind signatures for untraceable payments. In David Chaum and Ronald L. Rivest, editors, *Advances in Cryptology – CRYPTO 1982*, pages 199–203. Plemum Publishing, 1983. (Cited on page 51)

[Cha85] David Chaum. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM*, 28:1030–1044, October 1985. (Cited on page 5)

[Che00] Z. Chen. *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*. Java Series. Addison-Wesley, 2000. (Cited on pages 7 and 29)

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer Berlin / Heidelberg, May 2001. (Cited on pages 6 and 51)

[CL03] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer Verlag, 2003. (Cited on pages 6, 15, and 51)

[CL04] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72, 2004. (Cited on pages 20 and 68)

[Cop97] Don Coppersmith. Small solutions to polynomial equations, and low exponent rsa vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997. (Cited on page 14)

[Cor11a] Francisco Corella. Pros and cons of Idemix for NSTIC. `http://pomcor.com/2011/10/10/pros-and-cons-of-idemix-for-nstic/`, November 2011. (Cited on page 68)

[Cor11b] Francisco Corella. Pros and cons of U-Prove for NSTIC. `http://pomcor.com/2011/10/04/pros-and-cons-of-u-prove-for-nstic/`, November 2011. (Cited on page 68)

[CP93] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science (LNCS)*, pages 89–105. Springer Berlin Heidelberg, 1993. (Cited on page 25)

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In Jr. Kaliski, BurtonS., editor, *Advances in Cryptology - CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer Berlin Heidelberg, 1997. (Cited on page 21)

[CvH02] Jan Camenisch and Els van Herreweghen. Design and implementation of the idemix anonymous credential system. In *CCS 2002*, pages 21–30. ACM, November 2002. (Cited on page 51)

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976. (Cited on page 15)

[dlPHV14] Antonio de la Piedra, Jaap-Henk Hoepman, and Pim Vullers. Towards a full-featured implementation of attribute-based credentials on smart cards. In *Cryptology and Network Security - CANS2014*, Lecture Notes in Computer Science (LNCS). Springer-Verlag, October 2014. (to appear). (Cited on page 66)

[ECR09] ECRYPTII. Yearly report on algorithms and keysizes (2008-2009). Technical Report D.SPA.7, European Network of Excellence in Cryptology II (ECRYPTII), 2009. (Cited on page 34)

[EG85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology - CRYPTO 1984*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc. (Cited on pages 15, 16, and 17)

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew Odlyzko, editor, *Advances in Cryptology – CRYPTO 1986*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987. (Cited on pages 23 and 37)

[FST10] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, 2010. (Cited on page 20)

[Hås86] Johan Håstad. On using rsa with low exponent in a public key network. In *Advances in Cryptology - CRYPTO 1985*, pages 403–408, London, UK, UK, 1986. Springer-Verlag. (Cited on page 14)

[HJV10] Jaap-Henk Hoepman, Bart Jacobs, and Pim Vullers. Privacy and security issues in e-ticketing – Optimisation of smart card-based attribute-proving. In Veronique Cortier, Mark Ryan, and Vitaly Shmatikov, editors, *Workshop on Foundations of Security and Privacy, FCS-PrivMod*

*2010, Edinburgh, UK, July 14-15, 2010. Proceedings*, July 2010. (informal). (Cited on pages 5, 10, and 25)

[HMS09] D. Hankerson, A. Menezes, and M. Scott. Software implementation of pairings. In M. Joye and G. Neven, editors, *Identity-Based Cryptography*, volume 2 of *CIS*, pages 188–206. IOS Press, 2009. (Cited on page 30)

[IBM11] IBM Research Zürich Security team. Specification of the Identity Mixer cryptographic library, version 2.3.3. Technical report, IBM Research, Zürich, June 2011. (Cited on page 57)

[IBM12] IBM Research Zürich Security team. Specification of the Identity Mixer cryptographic library, version 2.3.4. Technical report, IBM Research, Zürich, February 2012. (Cited on pages 4, 6, 51, 52, 57, and 61)

[IEE00] IEEE standard specifications for public-key cryptography. *IEEE Std 1363-2000*, 2000. (Cited on page 31)

[ISO05] *ISO 7816-4 Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange.* ISO, Geneva, Switzerland, 2005. (Cited on page 6)

[ISO08] Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks, 2008. (Cited on page 25)

[Jou04] A. Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, 2004. (Cited on page 20)

[Lap12] Jorn Lapon. *Anonymous Credential Systems: From Theory Towards Practice.* PhD thesis, KU Leuven, Belgium, July 2012. De Decker, Bart (supervisor), Naessens, Vincent (cosupervisor). (Cited on page 68)

[LKdDN11] Jorn Lapon, Markulf Kohlweiss, Bart de Decker, and Vincent Naessens. Analysis of revocation strategies for anonymous Idemix credentials. In *Lecture Notes in Computer Science,*, pages 3–17. Springer, 2011. (Cited on page 68)

[Lys02] Anna A. Lysyanskaya. *Signature schemes and applications to cryptographic protocol design.* PhD thesis, Massachusetts Institute of Technology, September 2002. (Cited on page 51)

[MIR12] MULTOS implementation report. Technical Report MAO-DOC-TEC-010 v2.4, MAOSCO Limited, 2012. (Cited on pages 8 and 9)

[MV11] Wojciech Mostowski and Pim Vullers. Efficient U-Prove implementation for anonymous credentials on smart cards. In George Kesidis and Haining Wang, editors, *SecureComm 2011*, volume 96 of *LNICST*, pages 243–260. Springer-Verlag, 2011. (Cited on pages 5 and 11)

[NXP09] NXP Semiconductors. Smart solutions for smart services (z-card 2009). NXP Literature, Document 75016728, 2009. (Cited on page 8)

[Oka93] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer, 1993. (Cited on page 28)

[Paq11a] Christian Paquin. U-Prove cryptographic specification v1.1. Technical report, Microsoft Corporation, February 2011. (Cited on pages 45 and 70)

[Paq11b] Christian Paquin. U-Prove cryptographic test vectors v1.1. Technical report, Microsoft Corporation, February 2011. (Cited on page 47)

[Paq11c] Christian Paquin. U-Prove technology overview v1.1. Technical report, Microsoft Corporation, February 2011. (Cited on pages 45 and 69)

[Poi00] David Pointcheval. The composite discrete logarithm and secure authentication. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, volume 1751 of *Lecture Notes in Computer Science*, pages 113–128. Springer Berlin Heidelberg, 2000. (Cited on page 23)

[PS96] David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT96*, volume 1163 of *Lecture Notes in Computer Science*, pages 252–265. Springer Berlin Heidelberg, 1996. (Cited on page 38)

[PZ13] Christian Paquin and Greg Zaverucha. U-Prove cryptographic specification v1.1 revision 3. Technical report, Microsoft Corporation, December 2013. (Cited on pages 5, 37, and 70)

[Rog11] Pieter Rogaar. Attributes and tokens in U-Prove: Interval proofs and use cases. Master's thesis, Radboud University Nijmegen, The Netherlands, August 2011. (Cited on page 4)

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978. (Cited on page 13)

[RSA12] RSA Laboratories. PKCS #1 v2.2: RSA cryptography specifications, October 2012. (Cited on pages 8 and 14)

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989. (Cited on page 37)

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991. (Cited on pages 21 and 37)

[SGPV09] M. Sterckx, B. Gierlichs, B. Preneel, and I. Verbauwhede. Efficient implementation of anonymous credentials on Java Card smart cards. In *Information Forensics and Security – WIFS 2009*, pages 106–110. IEEE, 2009. (Cited on pages 6, 8, 35, 36, 61, 64, 65, and 69)

[Sun06a] Sun Microsystems, Inc. *Java Card 2.2.2 Application Programming Interface Specification*, March 2006. (Cited on page 31)

[Sun06b] Sun Microsystems, Inc. *Java Card 2.2.2 Virtual Machine Specification*, March 2006. (Cited on page 7)

[TJ09] Hendrik Tews and Bart Jacobs. Performance issues of selective disclosure and blinded issuing protocols on Java Card. In Olivier Markowitch, Angelos Bilas, Jaap-Henk Hoepman, Chris Mitchell, and Jean-Jacques Quisquater, editors, *WISTP 2009*, volume 5746 of *LNCS*, pages 95–111. Springer-Verlag, September 2009. (Cited on pages 5, 6, 32, 35, 36, 37, and 69)

[Tru07] Trusted Computing Group. TPM main specification version 1.2 rev. 116. `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`, July 2007. (Cited on page 6)

[VA13] Pim Vullers and Gergely Alpár. Efficient selective disclosure on smart cards using Idemix. In Simone Fischer-Hübner and Elisabeth de Leeuw, editors, *3rd IFIP WG 11.6 Working Conference on Policies and Research in Identity Management, IDMAN 2013, London, UK, April 8-9, 2013. Proceedings*, volume 396 of *IFIP Advances in Information and Communication Technology*, pages 53–67. Springer-Verlag, April 2013. (Cited on pages 6, 11, and 51)

[Ver01] Eric R. Verheul. Self-blindable credential certificates from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science (LNCS)*, pages 533–551. Springer Berlin Heidelberg, 2001. (Cited on pages 5, 25, 26, 27, and 28)

[Ver09] F. Vercauteren. Pairings on elliptic curves. In M. Joye and G. Neven, editors, *Identity-Based Cryptography*, volume 2 of *CIS*, pages 13–30. IOS Press, 2009. (Cited on page 20)

# Index

# Curriculum Vitae

## Pim Vullers

**June 20, 1986** Born in Venlo, The Netherlands

**September 1998 - August 2004** Pre-university secondary education (VWO)
*Atheneum*, Nature and Technology profile
Stedelijk Lyceum Roermond

**September 2004 - August 2007** Bachelor of Science
*Computer Science and Engineering*
Eindhoven University of Technology

**September 2007 - August 2009** Master of Science
*Computer Science and Engineering*, Information Security Technology track
Eindhoven University of Technology, Kerckhoffs Institute[7]

**September 2009 - February 2014** PhD
*Digital Security*
Radboud University Nijmegen

**March 2014 -**
*Security Engineer*
NXP Semiconductors

[7]The Kerckhoffs Institute is a cooperation of the Eindhoven University of Technology, the University of Twente and the Radboud University Nijmegen.

## Titles in the IPA Dissertation Series since 2008

**W. Pieters**. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

**A.L. de Groot**. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

**M. Bruntink**. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

**A.M. Marin**. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

**N.C.W.M. Braspenning**. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

**M. Bravenboer**. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates*. Faculty of Science, UU. 2008-06

**M. Torabi Dashti**. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

**I.S.M. de Jong**. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08

**I. Hasuo**. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

**L.G.W.A. Cleophas**. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

**I.S. Zapreev**. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

**M. Farshi**. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

**G. Gulesir**. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

**F.D. Garcia**. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathematics and Computer Science, RU. 2008-14

**P. E. A. Dürr**. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

**E.M. Bortnik**. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

**R.H. Mak**. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

**M. van der Horst**. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

**C.M. Gray**. *Algorithms for Fat Objects: Decompositions and Applications.*

*for Prime Time.* Faculty of Science, UU. 2009-9

**K.R. Olmos Joffré**. *Strategies for Context Sensitive Program Transformation.* Faculty of Science, UU. 2009-10

**J.A.G.M. van den Berg**. *Reasoning about Java programs in PVS using JML.* Faculty of Science, Mathematics and Computer Science, RU. 2009-11

**M.G. Khatib**. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

**S.G.M. Cornelissen**. *Evaluating Dynamic Analysis Techniques for Program Comprehension.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

**D. Bolzoni**. *Revisiting Anomaly-based Network Intrusion Detection Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

**H.L. Jonker**. *Security Matters: Privacy in Voting and Fairness in Digital Exchange.* Faculty of Mathematics and Computer Science, TU/e. 2009-15

**M.R. Czenko**. *TuLiP - Reshaping Trust Management.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

**T. Chen**. *Clocks, Dice and Processes.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

**C. Kaliszyk**. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web.* Faculty of Science, Mathematics and Computer Science, RU. 2009-18

**R.S.S. O'Connor**. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory.* Faculty of Science, Mathematics and Computer Science, RU. 2009-19

**B. Ploeger**. *Improved Verification Methods for Concurrent Systems.* Faculty of Mathematics and Computer Science, TU/e. 2009-20

**T. Han**. *Diagnosis, Synthesis and Analysis of Probabilistic Models.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

**R. Li**. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis.* Faculty of Mathematics and Natural Sciences, UL. 2009-22

**J.H.P. Kwisthout**. *The Computational Complexity of Probabilistic Networks.* Faculty of Science, UU. 2009-23

**T.K. Cocx**. *Algorithmic Tools for Data-Oriented Law Enforcement.* Faculty of Mathematics and Natural Sciences, UL. 2009-24

**A.I. Baars**. *Embedded Compilers.* Faculty of Science, UU. 2009-25

**M.A.C. Dekker**. *Flexible Access Control for Dynamic Collaborative Environments.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

**J.F.J. Laros**. *Metrics and Visualisation for Crime Analysis and Genomics.* Faculty of Mathematics and Natural Sciences, UL. 2009-27

**C.J. Boogerd**. *Focusing Automatic Code Inspections.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

**M.R. Neuhäußer**. *Model Checking Nondeterministic and Randomly Timed*

*Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

**J. Endrullis**. *Termination and Productivity.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

**T. Staijen**. *Graph-Based Specification and Verification for Aspect-Oriented Languages.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

**Y. Wang**. *Epistemic Modelling and Protocol Dynamics.* Faculty of Science, UvA. 2010-05

**J.K. Berendsen**. *Abstraction, Prices and Probability in Model Checking Timed Automata.* Faculty of Science, Mathematics and Computer Science, RU. 2010-06

**A. Nugroho**. *The Effects of UML Modeling on the Quality of Software.* Faculty of Mathematics and Natural Sciences, UL. 2010-07

**A. Silva**. *Kleene Coalgebra.* Faculty of Science, Mathematics and Computer Science, RU. 2010-08

**J.S. de Bruin**. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications.* Faculty of Mathematics and Natural Sciences, UL. 2010-09

**D. Costa**. *Formal Models for Component Connectors.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

**M.M. Jaghoori**. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services.* Faculty of Mathematics and Natural Sciences, UL. 2010-11

**R. Bakhshi**. *Gossiping Models: Formal Analysis of Epidemic Protocols.* Faculty of Sciences, Department of Computer Science, VUA. 2011-01

**B.J. Arnoldus**. *An Illumination of the Template Enigma: Software Code Generation with Templates.* Faculty of Mathematics and Computer Science, TU/e. 2011-02

**E. Zambon**. *Towards Optimal IT Availability Planning: Methods and Tools.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

**L. Astefanoaei**. *An Executable Theory of Multi-Agent Systems Refinement.* Faculty of Mathematics and Natural Sciences, UL. 2011-04

**J. Proença**. *Synchronous coordination of distributed components.* Faculty of Mathematics and Natural Sciences, UL. 2011-05

**A. Moralı**. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

**M. van der Bijl**. *On changing models in Model-Based Testing.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

**C. Krause**. *Reconfigurable Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-08

**M.E. Andrés**. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2011-09

**M. Atif**. *Formal Modeling and Verification of Distributed Failure Detectors.* Faculty of Mathematics and Computer Science, TU/e. 2011-10

**P.J.A. van Tilburg**. *From Computability to Executability – A process-theoretic view on automata theory.* Faculty of Mathematics and Computer Science, TU/e. 2011-11

**Z. Protic**. *Configuration management for models: Generic methods for model comparison and model co-evolution.* Faculty of Mathematics and Computer Science, TU/e. 2011-12

**S. Georgievska**. *Probability and Hiding in Concurrent Processes.* Faculty of Mathematics and Computer Science, TU/e. 2011-13

**S. Malakuti**. *Event Composition Model: Achieving Naturalness in Runtime Enforcement.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

**M. Raffelsieper**. *Cell Libraries and Verification.* Faculty of Mathematics and Computer Science, TU/e. 2011-15

**C.P. Tsirogiannis**. *Analysis of Flow and Visibility on Triangulated Terrains.* Faculty of Mathematics and Computer Science, TU/e. 2011-16

**Y.-J. Moon**. *Stochastic Models for Quality of Service of Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-17

**R. Middelkoop**. *Capturing and Exploiting Abstract Views of States in OO Verification.* Faculty of Mathematics and Computer Science, TU/e. 2011-18

**M.F. van Amstel**. *Assessing and Improving the Quality of Model Transformations.* Faculty of Mathematics and Computer Science, TU/e. 2011-19

**A.N. Tamalet**. *Towards Correct Programs in Practice.* Faculty of Science, Mathematics and Computer Science, RU. 2011-20

**H.J.S. Basten**. *Ambiguity Detection for Programming Language Grammars.* Faculty of Science, UvA. 2011-21

**M. Izadi**. *Model Checking of Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-22

**L.C.L. Kats**. *Building Blocks for Language Workbenches.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2011-23

**S. Kemper**. *Modelling and Analysis of Real-Time Coordination Patterns.* Faculty of Mathematics and Natural Sciences, UL. 2011-24

**J. Wang**. *Spiking Neural P Systems.* Faculty of Mathematics and Natural Sciences, UL. 2011-25

**A. Khosravi**. *Optimal Geometric Data Structures.* Faculty of Mathematics and Computer Science, TU/e. 2012-01

**A. Middelkoop**. *Inference of Program Properties with Attribute Grammars, Revisited.* Faculty of Science, UU. 2012-02

**Z. Hemel**. *Methods and Techniques for the Design and Implementation of Domain-Specific Languages.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03

**T. Dimkov**. *Alignment of Organizational Security Policies: Theory and Practice.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

**S. Sedghi**. *Towards Provably Secure Efficiently Searchable Encryption.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

**F. Heidarian Dehkordi**. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference.* Faculty of Science, Mathematics and Computer Science, RU. 2012-06

**K. Verbeek**. *Algorithms for Cartographic Visualization.* Faculty of Mathematics and Computer Science, TU/e. 2012-07

**D.E. Nadales Agut**. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation.* Faculty of Mechanical Engineering, TU/e. 2012-08

**H. Rahmani**. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2012-09

**S.D. Vermolen**. *Software Language Evolution.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

**L.J.P. Engelen**. *From Napkin Sketches to Reliable Software.* Faculty of Mathematics and Computer Science, TU/e. 2012-11

**F.P.M. Stappers**. *Bridging Formal Models – An Engineering Perspective.* Faculty of Mathematics and Computer Science, TU/e. 2012-12

**W. Heijstek**. *Software Architecture Design in Global and Model-Centric Software Development.* Faculty of Mathematics and Natural Sciences, UL. 2012-13

**C. Kop**. *Higher Order Termination.* Faculty of Sciences, Department of Computer Science, VUA. 2012-14

**A. Osaiweran**. *Formal Development of Control Software in the Medical Systems Domain.* Faculty of Mathematics and Computer Science, TU/e. 2012-15

**W. Kuijper**. *Compositional Synthesis of Safety Controllers.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16

**H. Beohar**. *Refinement of Communication and States in Models of Embedded Systems.* Faculty of Mathematics and Computer Science, TU/e. 2013-01

**G. Igna**. *Performance Analysis of Real-Time Task Systems using Timed Automata.* Faculty of Science, Mathematics and Computer Science, RU. 2013-02

**E. Zambon**. *Abstract Graph Transformation – Theory and Practice.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

**B. Lijnse**. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications.* Faculty of Science, Mathematics and Computer Science, RU. 2013-04

**G.T. de Koning Gans**. *Outsmarting Smart Cards.* Faculty of Science, Mathematics and Computer Science, RU. 2013-05

**M.S. Greiler**. *Test Suite Comprehension for Modular and Dynamic Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-06

**L.E. Mamane**. *Interactive mathematical documents: creation and presentation.* Faculty of Science, Mathematics and Computer Science, RU. 2013-07

**M.M.H.P. van den Heuvel**. *Composition and synchronization of real-time components upon one processor.* Faculty of Mathematics and Computer Science, TU/e. 2013-08

**J. Businge**. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins.* Faculty of Mathematics and Computer Science, TU/e. 2013-09