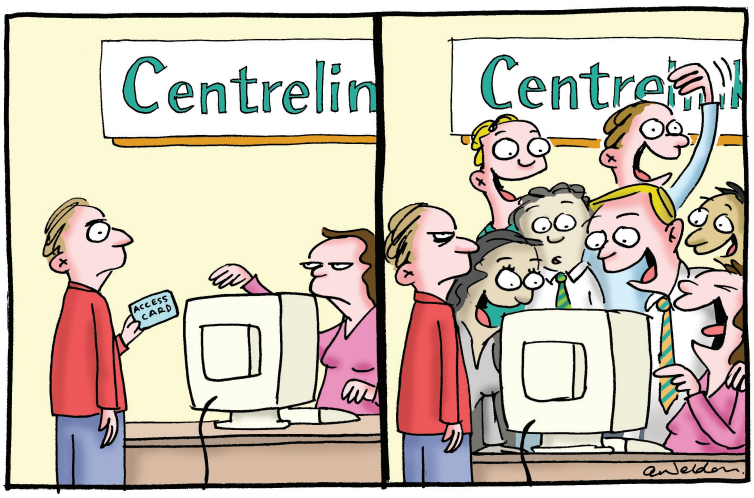
university of groningen / faculty of mathematics and natural sciences

TNO

Providing unlinkability of transactions with a single token in U-Prove

Erik Weitenberg



Master's Thesis in Mathematics

July 2012

Providing unlinkability of transactions with a single token in U-Prove

Summary

Using the U-Prove system originally conceived by Stefan Brands [2000], one can obtain credentials from a central authority, and partially or completely disclose them to relying parties. The user's privacy is guaranteed as long as he does not show any credential more than once. However, this requirement forces a privacy-conscious user to request and store many copies of essentially the same credentials.

We present a modified set of protocols intended to make linking the different times a credential was shown infeasible, while retaining unlinkability between the issuing and showing phases.

Master's Thesis in Mathematics
Author: Erik Weitenberg
Supervisors: Jaap-Henk Hoepman, Jaap Top
Date: July 2012

Johann Bernoulli Institute
P.O. Box 407
9700 AK Groningen
The Netherlands

Preface

This thesis is the final part of my education to obtain a Master's degree in mathematics at the University of Groningen. It is also the result of a nine month internship at TNO in Groningen in the Security group.

This project would not have been possible without the support of many people. I would like to express my gratitude towards Jaap-Henk Hoepman and Jaap Top, my supervisors, for introducing me to the field of cryptology and for their patient and enthusiastic help during my research project.

Furthermore, I would like to thank my colleagues at TNO for the wonderful time I have had during my internship there, and especially Wouter Lueks and Gergely Alpár for their guidance and help in writing this thesis, and for the interesting discussions we had about their research.

Finally, I wish to thank my friends and family for their continuing support, strength and encouragement throughout the duration of my studies.

Contents

1

Introduction

In recent years, many advances have been made in technology intented for use by law enforcement, such as full-body scanners and automated aggregation of all kinds of data about citizens. This leaves many people concerned about their privacy. Perhaps rightfully so: according to research by *Bits of Freedom*, citizens' privacy takes a back seat as far as the Dutch police is concerned.[1]

On the other hand, many claim that the privacy concern is indeed second to the need to promote efficiency and public safety, and that the measures constitute only a minor violation of privacy. Unfortunately, this argument sometimes underestimates the amount of information you can gain by collecting very little data: for example, in the United States, 87% of citizens could in 2002 be identified by just the combination of their date of birth, gender and zip-code. [Sweeney, 2002]

Instead of trying to decide which of these two needs has to give way to the other, it would be nice if we could accomodate both of them. Current systems are often more powerful than they need to be, and rely on their operators to follow the rules (for example, to delete certain data that doesn't need to be kept). A more desirable system might collect no identifying data about anyone, except for the data it needs to function correctly. This effort is sometimes reffered to as *privacy by design*, and this thesis is part of that effort.

[1] Research findings and sources (in Dutch): `www.bof.nl/2012/07/04/persbericht-politie-overtreedt-op-grote-schaal-wet-bij-gegevensbescherming`

1.1 Credentials

Suppose you were to go to the store to buy delicious rum. In most parts of the world, the store owner is obliged to ask you to prove that you are older than 18 years old, or maybe even 21. You can do this by showing him your drivers' licence or perhaps your passport; if you do so, the salesman will indeed believe that you are old enough.

You could, of course, also try to just say "I am 19 years old" very convincingly, but this doesn't often work. This is natural to most of us: the important thing is not just the message 'more than 18 years old', but also the one attesting it (in this case, the government). This is why you normally use a drivers' licence: it's hard to falsify, and genuine ones are printed by the government and contain the holder's date of birth.

We often refer to this combination of a statement and a way to verify its authenticity as a *credential*. Credentials are everywhere. Your passport and drivers' licence are common examples, and so is your high school diploma, a combination of a username and password to your e-mail inbox, or even the key to your house.

In this work, we will mainly concern ourselves with credentials that are verified automatically, since a computer is in a much better position than a human to remember all credentials it sees. A popular example is found on the *smartcard*, a small card that looks like a credit card, but contains a tiny computer, capable of storing some information and performing some computations. These can store credentials and reproduce them when held to a card reader. They are currently used to pay for public transit, for example in the Netherlands (the OV-chipkaart) and the city of London (the Oyster card). Many modern passports contain chips as well.

While much of our discussion is also applicable to mobile phones or personal computers, smartcards present an additional challenge. Their limited processing power and memory require that handling credentials is quick and doesn't require much memory.

1.2 U-Prove

This thesis is concerned with methods of showing credentials to others – for example, to a shopkeeper, in order to buy restricted goods. Doing that normally does not impact your privacy very much, since the shopkeeper probably won't remember you. Using smart cards for this complicates things a little: you can't just say "here's my passport data," since anyone can then copy that data and pretend to be you. To solve this problem, methods exist

to prove to someone that you have a passport with certain information on it, and also that you are actually the person the passport was issued to.

One such method, and the main subject of this thesis, is U-Prove. Designed by Brands [2000], it provides sophisticated ways to reveal credentials, even partially, and to prove statements about certain types of information without completely revealing it (for example, "my date of birth is more than 18 years ago"). Also, the shopkeeper can't find out if you are the same person who bought rum just yesterday or not, even if he is secretly a government agent and has copies of all credentials ever made.

Unfortunately, privacy has its price. Currently, the guarantee that nobody knows what you do with your credential only holds if you request many credentials, and discard each one after showing it to someone. With smart cards, credentials are just files, and handling multiple credentials is not very difficult; still, it's cumbersome, because today's smart cards do not have a very large amount of storage, and nobody wants to go out and get new credentials every week.

The cause of this problem is that, in U-Prove, the credential you get from the government is issued using a *blind signature* – that is, the government agent doesn't see the final credential, but instead signs an intermediate form, which the recipient can then transform into the final credential on her own. When showing it, she hands over the entire credential and proves it is indeed hers. Of course, the shopkeeper can just save the credential with the receipt of your purchase, which is why you shouldn't use it twice if you're concerned about your privacy.

1.3 Problem statement

Currently, using U-Prove in a privacy-conscious way implies discarding a credential after using it just once, as discussed above. We would like to modify the U-Prove system such that it no longer has this requirement, but does retain the current advantages of selective disclosure of credentials and the ability to prove compound statements about them. Of course, this modified system should also guarantee that it is still not possible to connect different transactions made using the same credential, as is currently the case.

1.4 TNO

This thesis is the result of an internship at TNO, which is short for *Nederlandse Organisatie voor Toegepast Natuurwetenschappelijk Onderzoek*. TNO is a

research organisation which aims to develop practical knowledge and expertise with which to assist and advise companies and governments.

During my internship at TNO, I was part of the security group, which studies the security and privacy aspects of digital systems. This work is very varied; some of my collegues do practical research on smartcard software, while others perform risk assessments, and audits for other companies, or help create new security policies.

Much of TNO's work is done in response to customer orders, but to maintain a useful knowledge base, it also needs to do research that is not the direct result of an external request. This thesis is part of that effort.

1.5 Reading guide

This thesis is intended to be readable by anyone with a background in mathematics. To this end, chapter 2 provides a brief introduction to cryptography, with references to more in-depth material. For those with a general background in cryptography, chapter 3 elaborates on interactive proofs of knowledge and chapter 4 details the precise workings of two U-Prove protocols for issuing and showing credentials. Chapter 5 contains our proposed modifications of the U-Prove system, and a discussion of their security. Finally, chapter 6 concludes this thesis with a summary and suggestions for further research.

2

Basic cryptography

Cryptography, in general, deals with the problem of sending a message to someone, in such a way that no-one but the intended recipient can read it. The problem itself has existed for quite a while, and many solutions have been invented, used and broken over the years. As an introduction, we will briefly look at a few historical methods of secret keeping. We'll then introduce a category of methods called public-key cryptography, which is quite popular today. Finally, we present a brief introduction to cryptography using elliptic curves.

Readers who are interested in exploring these subjects in (much) more detail are advised to read [Smart, 2003].

2.1 Secrets and eavesdroppers

Imagine two people, called Alice and Bob for mostly historical reasons. Alice would like to send a secret message to Bob. However, they live far apart and can only send their message using the postal service. A person called Eve works at the post office, and she will open the message secretly and try very hard to read it.

To prevent Eve from succeeding, Alice can try to use a code language to change her message into something Eve will never understand. One way of doing this is by replacing every letter with another letter, for example, `A` becomes `C`, `B` becomes `D` and so on. However, given a long message, it's very conceivable that Eve will pick up on the trick. Alice can make it a bit harder

by using a more random table of replacements, like the following:

$$\begin{bmatrix} \texttt{A} & \texttt{B} & \texttt{C} & \texttt{D} & \texttt{E} & \texttt{F} & \texttt{G} & \texttt{H} & \texttt{I} & \texttt{J} & \texttt{K} & \texttt{L} & \texttt{M} & \texttt{N} & \texttt{O} & \texttt{P} & \texttt{Q} & \texttt{R} & \texttt{S} & \texttt{T} & \texttt{U} & \texttt{V} & \texttt{W} & \texttt{X} & \texttt{Y} & \texttt{Z} \\ \texttt{D} & \texttt{N} & \texttt{R} & \texttt{S} & \texttt{O} & \texttt{C} & \texttt{G} & \texttt{Y} & \texttt{M} & \texttt{V} & \texttt{T} & \texttt{U} & \texttt{P} & \texttt{Q} & \texttt{B} & \texttt{H} & \texttt{X} & \texttt{L} & \texttt{E} & \texttt{F} & \texttt{A} & \texttt{J} & \texttt{I} & \texttt{Z} & \texttt{W} & \texttt{K} \end{bmatrix}$$

Still, if the message is long enough, Eve can find out what it says. In English, the letter `E` is used most often, followed by `T`, `A` and so on. By counting how often each letter appears in the secret message, she can probably recover enough columns in the table to understand the message completely.

Now, Alice can improve this system still: she could use five (or five hundred) tables of substitutions, encoding each subsequent letter using a different table. However, people have computers these days, and if Eve waits long enough she might be able to collect enough text to find out how many tables Alice has used, and what substitutions they contain.

The above is an example of a symmetric encryption: you need the same secret information (the key) to encrypt the message as you need to decrypt it. The symmetric encryption schemes used in the 'real' world are more advanced, but they share a common problem: for both parties to share the same secret key, they must have communicated securely in the past. Also, Alice cannot re-use the key she shares with Bob to exchange secret messages with Charlie, since Bob will be able to read those messages as well; hence, many more keys are needed as the number of people inceases.

This branch of cryptography is called *symmetric cryptography*. We will be avoiding it in favour of public-key cryptography, but if you're interested, Nigel Smart's *Cryptography: an Introduction* is a good place to start.

2.2 Public-key cryptography

In the previous example, both Alice and Bob had to know a secret key. This is not always feasible, and fortunately an alternative exists in the form of asymmetric encryption schemes. We will discuss two examples.

2.2.1 The ElGamal cryptosystem and CDH

All operations in the ElGamal system take place in a multiplicative group G, generated by an element g of prime order p. These parameters should be known to everyone in the system.

Alice first generates a private key, which is a number x chosen uniformly at random from $\mathbb{Z}/p\mathbb{Z}$. (We will henceforth denote such random choices by $\in_{\mathcal{R}}$.) She calculates her public key $h = g^x$, and makes sure everyone knows it.

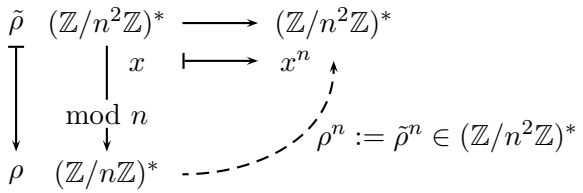


Figure 2.1: The kernel of the mod n map is the same as the kernel of the exponentiation map, i.e. $(1 + n\mathbb{Z})/(n^2\mathbb{Z})$.

Bob can now send Alice an encrypted message. He starts with a message m, which is an element of G, and Alice's public key h. First, he picks a random number $y \in \mathbb{Z}/p\mathbb{Z}$ and calculates the first part of the ciphertext, $c_1 = g^y$. The second part of the ciphertext is $c_2 = m \cdot h^y$. He sends (c_1, c_2) to Alice.

Alice assumes that the ciphertext she just received is made in the above way, and she knows x, but not y. Still, she can calculate $h^y = g^{xy}$ as c_1^x. She calculates the inverse of this element in G, and recovers the message by calculating $m = c_2 \cdot (c_1^x)^{-1}$.

Though it is very difficult to decrypt ElGamal-encrypted values without knowing the secret key – which we will prove in section 2.5 – it does have one particular weakness called malleability. Suppose Bob has sent Alice a message $(g^y, m \cdot h^y)$, but Eve intercepts it and multiplies the second part of the ciphertext by 2 before sending it on to Alice. Now when Alice decrypts the message, she will get $2m$ instead of m.

2.2.2 The Paillier cryptosystem

In the next chapters, we will use a slightly more complicated system for excryption: the Paillier system [Paillier and Pointcheval, 1999]. It uses a group $G = \mathbb{Z}/n^2\mathbb{Z}$, where n is the product of two large primes p_1 and p_2. We also choose an element $g \in (\mathbb{Z}/n^2\mathbb{Z})^*$ such that n divides $\mathrm{ord}(g)$. To enable Bob to send her an encrypted message, Alice keeps the p_i secret, and broadcasts g and n.

To encrypt a message $m \in [0, n)$ for Alice, Bob calculates $[\![m]\!] := g^m \rho^n \mod n^2$, where $\rho \in_{\mathcal{R}} \mathbb{Z}/n\mathbb{Z}^*$. To do this, he lifts ρ to its lowest representative in $(\mathbb{Z}/n^2\mathbb{Z})^*$, see figure 2.1.

Again, Alice has a way to decrypt the ciphertext using her secret information: she calculates $\lambda = \mathrm{lcm}(p_1 - 1, p_2 - 1)$. Then

$$m = \frac{L(c^\lambda \mod n^2)}{L(g^\lambda \mod n^2)} \mod n,$$

where $L(x) := (x-1)/n$. To understand that this works, observe that $g^\lambda \in \ker(\bmod n)$, so it can be written as $1+an$ for some number a, which is unique up to multiples of n. In this group, $(1+an)^k \equiv 1+akn$, hence $g^{\lambda m} \equiv 1+amn \bmod n^2$.

Since we also know $g^\lambda \equiv 1+an$, it is now possible to calculate m. It is a solution of the equation $anm \equiv g^{\lambda m} - 1 \mod n^2$, and therefore of

$$am \equiv \frac{g^{\lambda m}-1}{n} \mod n.$$

This equation has a unique solution $m \in [0, n)$ provided that a is a unit modulo n, which it is because n divides the order of g.

The advantage of Paillier encryption over ElGamal is that it is possible for Bob to perform calculations on the encrypted data without getting access to the plaintext. Two operations are possible:

- Addition: $[\![m_1]\!] \cdot [\![m_2]\!]$ decrypts to $m_1 + m_2 \mod n$;
- Multiplication by a scalar: $[\![m]\!]^k$ decrypts to $k \cdot m \mod n$.

This way, Alice can send some encrypted values to Bob, who performs some operation involving his secret data, and then sends the result back to Alice. Alice can decrypt and use the result, while Bob never sees Alice's secret data.

2.3 Signatures

Although it is nice to be able to prevent others from reading the messages you send, it is not the only thing you can do with cryptography. A second popular application is the digital signature, which allows you to 'sign' a message. The recipient of your message can check the signature, and if anyone has tampered with your message after you signed it, the signature will be invalid.

2.3.1 Hash functions

We will illustrate the concept using the ElGamal signature. This signature scheme needs a special kind of function called a *hash function*. An ideal hash function transforms its input into an entirely random number in a certain range or group, but when given the same input twice, it will produce the same output twice as well.

In reality, hash functions $\mathcal{H}$ can not behave entirely randomly; they use a deterministic algorithm to arrive at their answer. Their 'strength' depends on how difficult it is

1. to find two inputs m_1 and m_2 such that $\mathcal{H}(m_1) = \mathcal{H}(m_2)$ (collision-freedom), or
2. to find an input m, given a desired output h, with $\mathcal{H}(m) = h$ (pre-image resistance), or
3. to find an input m_2 given m_1 and h such that $\mathcal{H}(m_1) = \mathcal{H}(m_2) = h$ (second pre-image resistance).

In general, the output of a hash function is of a fixed size, while it accepts input of any size. The fact that the output of a hash function changes dramatically if the input is changed even slightly is often used to verify, for example, that a file has not been damaged while being sent over a network.

2.3.2 The ElGamal signature scheme

Suppose Alice sends a message m to Bob, and wants to sign it. Again, she randomly chooses a secret key x and calculates her public key $h = g^x$.

To sign the message, she generates a random number $y \in (0, p-1)$ with $\gcd(y, p-1) = 1$. With it, she computes $s_1 = g^y \pmod p$ and $s_2 = (\mathcal{H}(m) - xs_1)y^{-1} \pmod{p-1}$. In the event that $s_2 = 0$, Alice starts over. She sends the signature (s_1, s_2) to Bob along with the message.

Bob then verifies the signature by checking if $g^{\mathcal{H}(m)} \stackrel{?}{=} h^{s_1} s_1^{s_2} \pmod p$. If the signature is correct, the right hand side is equal to

$$\begin{aligned} h^{s_1} s_1^{s_2} &\equiv h^{g^y} g^{y(\mathcal{H}(m) - xs_1)y^{-1}} \\ &\equiv g^{xg^y} g^{\mathcal{H}(m) - xg^y} \\ &\equiv g^{\mathcal{H}(m)}. \end{aligned}$$

The ElGamal signature is believed to be secure as long as the hash function is secure, though no reduction to a complexity assumption is known [ElGamal, 1985].

2.4 Cryptography using elliptic curves

Until now, we have confined our discussion to groups of integers modulo n. While this makes for a simple discussion, there are other choices. A common one is the group of points on an elliptic curve. In this section, we give a short overview of this group and its advantages over $\mathbb{Z}/p\mathbb{Z}$. For a more thorough introduction, see Silverman and Tate [1992].

2.4.1 Generic elliptic curves

The elliptic curves we will study are the solution sets to equations that look like this:

$$y^2 = x^3 + ax + b. \tag{2.1}$$

We take a, b, x and y to be elements of a field K of characteristic[1] not equal to 2 or 3, such that the resulting curve is smooth. We will first consider the case where $K = \mathbb{Q}$, as this makes for nice pictures like the ones in figure 2.2; in the later chapters, K will usually be a prime field $\mathbb{Z}/p\mathbb{Z}$.

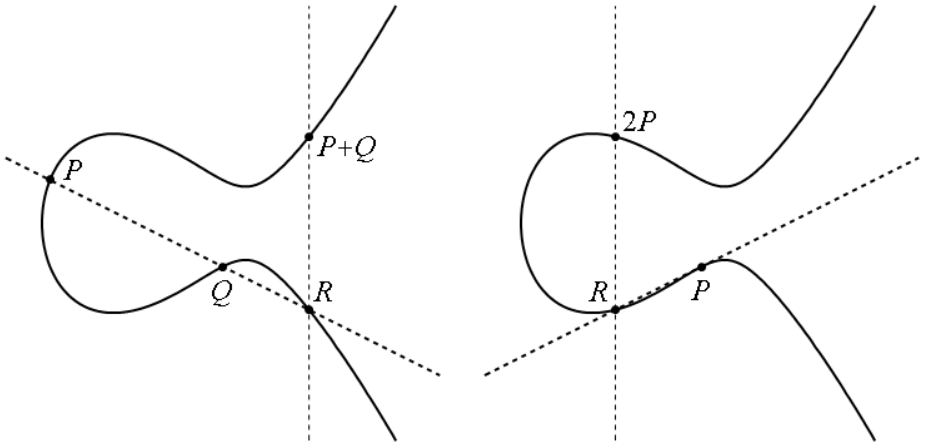
The group of points on an elliptic curve, $E(K)$, can then be defined as those points $(x, y) \in K^2$ that satisfy the equation, together with a point at infinity $\mathcal{O}$ which will act as the group's zero element.

The group operation, point addition, is based on the rule that if three points on an elliptic curve are collinear, their 'sum' is defined to be zero. The point at infinity is defined to lie on all vertical lines. With this in mind, we conclude that the additive inverse of a point $P = (x, y)$ must be the point $-P = (x, -y)$.

To 'add' two nonzero points $P, Q \in E(K)$, we

1. draw a line connecting the points P and Q,

[1] A field's characteristic is the smallest number of times you must add 1 to itself to obtain 0 in the field. If this is not possible, the characteristic is defined to be 0.



(a) An example of point addition. (b) An example of point doubling.

Figure 2.2: Point addition and doubling on the elliptic curve defined by $y^2 = x^3 - 7x + 10$.

2. find the third point of intersection R (counting multiplicity) of the line and E, and
3. keeping in mind that $P + Q + R = \mathcal{O}$, define $P + Q := -R$.

A remark about this procedure are in order: adding a point P to itself is not possible using this method. Instead of step 1, one should

1'. draw a line through P, tangent to the curve.

The group we have just created is in many cases infinite, and isomorphic to the direct sum of one or two finite cyclic groups ($\mathbb{Z}/n\mathbb{Z}$ for various n) and a number of copies of $\mathbb{Z}$: w

$$E(\mathbb{Q}) \simeq \mathbb{Z}/n_1\mathbb{Z} \oplus \mathbb{Z}/n_2\mathbb{Z} \oplus \mathbb{Z} \oplus \cdots \oplus \mathbb{Z}.$$

2.4.2 Explicit formulas for the group law

Having to construct points geometrically gets tedious, but fortunately it is possible to construct explicit formulas for point addition.

First, let's see what happens when we add two different points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ on the curve, assuming $P \neq -Q$. The line that connects them is given by

$$y = \lambda x + \mu, \quad \lambda = \frac{y_Q - y_P}{x_Q - x_P}, \quad \mu = y_P - \lambda x_P.$$

By substituting this for y into the elliptic curve equation 2.1, we get an equation in x:

$$x^3 - \lambda^2 x^2 + (a - 2\lambda\mu)x + (b - \mu^2) = 0.$$

The roots of this equation are precisely the coordinates x_P, x_Q and x_R, so we find

$$x^3 - \lambda^2 x^2 + (a - 2\lambda\mu)x + b - \mu^2 = (x - x_P)(x - x_Q)(x - x_R),$$

hence

$$x_R = \lambda^2 - x_P - x_Q \quad \text{and} \quad y_R = \lambda x_R + \mu.$$

Of course, since $P + Q = -R$, $y_{P+Q} = -y_R$.

If $P = Q$ (i.e. we are calculating $2P$), only the slope of the line changes, as it is now a tangent line to the curve:

$$\lambda = \frac{3x_P^2 + a}{2y_P}.$$

2.4.3 Elliptic curves over finite fields

For our purposes, it is more convenient to use finite fields instead of $\mathbb{Q}$. The resulting elliptic curve group then becomes finite as well; in fact, the elliptic curve groups $E(\mathbb{Z}/p\mathbb{Z})$ are all isomorphic to a cyclic group or the direct sum of two cyclic groups.

Example. The elliptic curve we considered earlier also exists over the finite field $\mathbb{Z}/7\mathbb{Z}$, where it is defined by the equation $y^2 = x^3+3$. By trying all values for x and y between 0 and 6, we find that twelve points in $\mathbb{Z}/7\mathbb{Z}^2$ satisfy this equation: $(1,2), (1,5), (2,2), (2,5), (3,3), (3,4), (4,2), (4,5), (5,3), (5,4), (6,3)$ and $(6,4)$. The elliptic curve group contains these points and the point at infinity, $\mathcal{O}$.

Of course, we can also use the group structure to find other points if we find one point more or less by chance. Suppose we know that $P = (1,2)$ is on the curve. Then we can use the formulas above to calculate

$$
\begin{array}{lll}
P = (1,2) & 6P = (5,3) & 10P = (2,5) \\
2P = (6,3) & 7P = (5,4) & 11P = (6,4) \\
3P = (2,2) & 8P = (3,4) & 12P = (1,5) \\
4P = (4,5) & 9P = (4,2) & 13P = \mathcal{O} \\
5P = (3,3) & &
\end{array}
$$

Therefore, the points on this curve form a group isomorphic to $\mathbb{Z}/13\mathbb{Z}$. See figure 2.3 for an illustration of the group's structure.

2.5 Proof techniques

The above example also illustrates what makes elliptic curves suitable for cryptography: the coordinates of these points are, at a glance, random. Given the point $(4,2)$ on the curve, it is not immediately obvious how many times you have to add P to itself to obtain $(4,2)$, and indeed, the difficulty of solving this problem for curves over even medium-sized fields is one of the things that makes elliptic curve cryptography possible.

In general, to prove security of cryptographic schemes, we will often use a special kind of axioms called complexity assumptions. These consist of a difficult problem like the one mentioned above, and the assertion that solving such a problem takes a very long time; to be precise, the time it takes grows exponentially as a function of the number of bits needed to express the size of the group we are using. The above problem is summarised in the following complexity assumption:

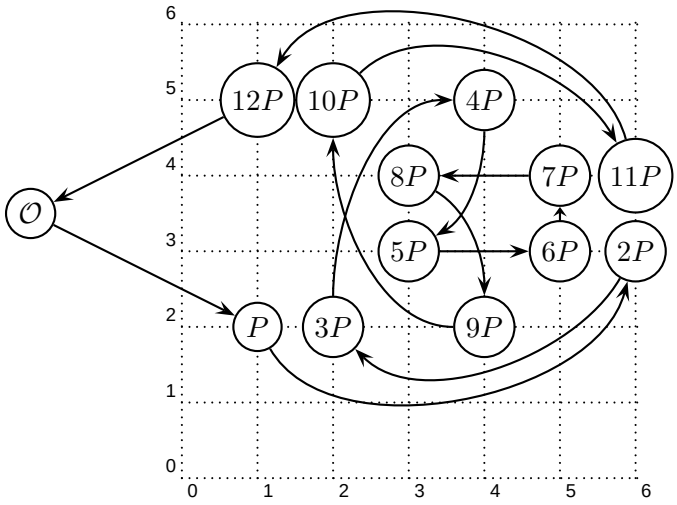


Figure 2.3: Illustration of the curve group of $y^2 = x^3 + 3$ over $(\mathbb{Z}/7\mathbb{Z})^2$. The arrows represent addition of P.

Complexity Assumption (Discrete Logarithm (DL))**.** Given an elliptic curve group $\mathbb{E} \subseteq E_{a,b}(\mathbb{Z}/p\mathbb{Z})$ with generator P and a random point $A \in \mathbb{E}$, it is difficult to calculate a such that $aP = A$.

In this case, the reason it takes so long to solve this problem is that you have to calculate aP for every possible value of a to see if it happens to be equal to A. Slightly faster algorithms exist, but they do not bring the required time down to a level we call 'fast', that is, polynomial time.

To prove that a certain cryptographic scheme is secure, we will often argue by reduction to absurdity: if someone can quickly break a given scheme, we can also make him solve a problem considered difficult by disguising it as, for example, an encrypted message. This type of proof is called a reduction, and the 'someone' is called the *adversary*.

To illustrate this method of proving, we will first discuss an example. We will also list some complexity assumptions to be used later on, and their relation to each other.

2.5.1 ElGamal encryption revisited

As an example, we will prove that it is difficult to reverse the ElGamal encryption scheme. The proof relies on the following complexity assumption about a group G:

Complexity Assumption (Computational Diffie-Hellman (CDH))**.** Given a group G with generator g and two elements $g^a, g^b \in G$, it is very difficult to compute g^{ab}.

Lemma 1. *Someone who can reverse the ElGamal encryption in polynomial time can also solve the Computational Diffie-Hellman problem in polynomial time.*

Proof. Suppose Eve can reverse the ElGamal encryption: given a generator g of a group G, a public key h and a ciphertext (c_1, c_2), she will produce a plaintext m.

As we attempt to solve the CDH problem, we are given g^a and g^b. We tell Eve that g^a is Alice's public key, and give her (g^b, g^z) for some random z as the ciphertext. Eve will proceed to give us a plaintext m; when she does we calculate g^z/m. This is the solution to the CDH problem, since

$$(c_1, c_2) = (g^b, g^z) = (g^b, mh^b) = (g^b, mg^{ab}).$$

Therefore, assuming Eve has correctly calculated m, her method can be used to solve the CDH problem as well. Hence it is at least as hard to reverse ElGamal encryption as it is to solve the CDH problem. □

2.5.2 The discrete-logarithm representation

In the following chapters, we will rely on one more assumption.

Complexity Assumption (Discrete-logarithm representation (DLREP))**.** Given a group $\mathbb{E}$ containing points $P_1, \ldots, P_n$ and A, it is hard to find a *discrete-log representation* $a_1, \ldots, a_n$ such that

$$A = \sum_{i=1}^{n} a_i P_i.$$

Proof by reduction to DL. Again, we assume that an adversary exists that can solve the above problem. She will help us solve the DL problem.

Suppose we are given an instance of the DL problem, that is, a group $\mathbb{E}$ with generator P and a random point A. We pick some random x_i and set $P_i = x_i P$ for $i = 1, \ldots, n$. We have the adversary give us a_i. Now,

$$\log_P A = \sum_{i=1}^{n} a_i x_i,$$

which solves the DL problem. □

3

Proofs of knowledge

Knowing a secret (for example a password, a secret key, etc.) is in general not very useful if you can't convince anyone else you know it; and if you just tell them the secret they will believe you, but the secret won't be very secret anymore. This is why many cryptographic protocols, including U-Prove, employ proofs of knowledge.

3.1 An example: where's Wally?[1]

Starting in 1987, the British illustrator Martin Handford published several books in a series called *Where's Wally?* (known in the United States as *Where's Waldo?*). They contain large illustrations of crowds, and if you look long enough you can find Wally, a man in a red-and-white striped shirt, somewhere in each illustration.

Now, suppose you've found Wally on a print and want to convince your friend you know where he is, without just pointing him out (since that would ruin the game). A possible way to do this is to find a large sheet of cardboard and cut a Wally-sized hole in the middle. Then, you can hold the print behind the cardboard sheet, so Wally is visible through the hole. Now, your friend sees Wally, but doesn't know where on the page he is.

Proofs of knowledge come in two varieties: interactive (like the Wally-proof) and non-interactive. The former requires that the prover and the verifier of the proof exchange messages in turn, the latter allows the prover to 'write

[1] This example is taken from, and discussed in much more detail in Naor et al. [1999].

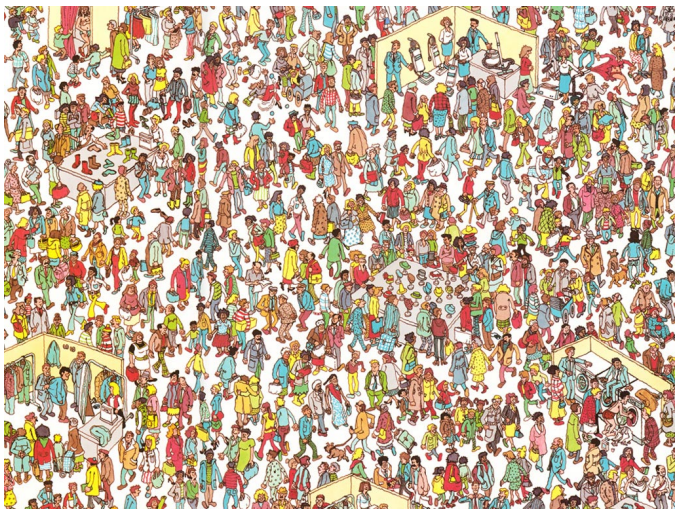


Figure 3.1: The Department Store illustration from one of Wally's books

down' the entire proof so it can be verified later. We will mainly concern ourselves with interactive proofs of knowledge.

Ideally, a proof of knowledge convinces the verifier that the prover knows a secret, but does not 'leak' any information. Also, like the Wally-proof, the verifier should not be able to use the proof to convince anyone else. We call a proof with these properties a *zero-knowledge proof of knowledge*.

3.2 The Schnorr proof of knowledge

As a first mathematical proof of knowledge, we will discuss the Schnorr proof of knowledge (POK). It allows us to prove, given a group $\mathbb{E}$ generated by a point P of order q, and a point $X \in \mathbb{E}$, that we know a number $x \in \mathbb{Z}/q\mathbb{Z}$ such that $xP = X$. Recall that it is difficult for the verifier to calculate x himself, since doing that requires him to solve the Discrete Logarithm problem in $\mathbb{E}$.

The proof of knowledge consists of a four-step protocol between the Prover and the Verifier. It works as follows:

Commitment First, the Prover generates a random number w in $\mathbb{Z}/q\mathbb{Z}$. She sends $W = wP$ to the Verifier. This random point on the curve is called the commitment.

Challenge The Verifier generates a random number γ in $\mathbb{Z}/q\mathbb{Z}$. He sends it to the Prover.

Response The Prover now calculates $r = \gamma x + w$ and sends this value to the Verifier.

Verification The Verifier now checks that $W = rP - \gamma X$. If so, he believes the Prover indeed knows x.

This protocol is summarised below. We will often lay protocols out in tables like these; each line contains one move made by one of the participants. These should be read like a computer program; the moves are made strictly in the order described, and no move begins before the previous one has completed.

Protocol 3.1: **Schnorr's proof of knowledge**

Common information: group $\mathbb{E}$ generated by point P with order q, and a multiple X of P.
Private information for the Prover: the number $x \in \mathbb{Z}/q\mathbb{Z}^*$ such that $xP = X$.

Prover		**Verifier**	**Comments**
select $w \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$			
send wP	$\longrightarrow$	into W	*Commitment*
		select $\gamma \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$	
into γ	$\longleftarrow$	send γ	*Challenge*
send $\gamma x + w$	$\longrightarrow$	into r	*Response*
		verify $W \stackrel{?}{=} rP - \gamma X$	*Verification*

Note that it is quite possible to use this protocol as a means of identifying someone: if X is Alice's public key, she can use a Schnorr proof of knowledge to prove she knows the corresponding private key.

3.2.1 Proofs of security

The Schnorr proof of knowledge, while simple, is quite safe, so we can use it to introduce the different aspects of security. The Schnorr proof of knowledge does assume the Verifier is *honest*; that is to say, he follows the protocol correctly.

Completeness A Prover who follows the protocol correctly and knows x will be able to convince an honest Verifier of this fact.

Soundness Only with negligible probability can a cheating Prover convince an honest Verifier that she knows x, even though she really doesn't.

Honest-verifier zero knowledge The only thing an honest Verifier learns from an execution of the protocol is that the prover knows x. In particular, he gains no evidence with which to convince anyone else.

Proposition 1. *The Schnorr proof of knowledge is complete.*

Proof. The Verifier checks if $W = rP - \gamma X$. But $W = wP$, $X = xP$ and $r = \gamma x + w$, so this comes down to verifying that

$$wP = (\gamma x + w)P - \gamma xP,$$

which is obviously true. □

Proposition 2. *The Schnorr proof of knowledge is sound.*

To prove soundness, we require a new proof technique:

Extraction We assume, as is often the case, that all participants in the protocol are computers running algorithms that tell them how to engage in the protocol. This means we can stop, restart or rewind them as well. This allows us to extract certain valuable information from an adversary which has the ability to cheat at our proof of knowledge.

Proof. Suppose we are given a Prover-algorithm that has a good chance of completing a successful Schnorr proof without knowing x for more than one possible challenge sent by the Verifier. Since we are assuming the Prover can be stopped and resumed at will, we stop it after it has sent its commitment. By restarting it twice from this state, but giving it different challenges, we have a non-negligible probability to end up with two tuples (W, γ, r) and (W, γ', r').

Assuming the Verifier accepts both proofs, we can now calculate x:

$$x := \frac{r - r'}{\gamma - \gamma'}.$$

This violates the DL assumption: using the Prover, we now have a way to compute the discrete logarithm of X with non-negligible probability. □

Of course, this assumes the Verifier chooses his challenges randomly; if the Prover can predict γ, she can easily fool the Verifier by choosing any r and setting $W := rP - \gamma X$.

Proposition 3. *The Schnorr proof of knowledge is honest-verifier zero-knowledge.*

Proof. To show this, we argue that the Verifier can create a valid transcript without knowledge of x or the Prover's help, and no one will be able to distinguish a fake transcript from a real one.

To simulate a Schnorr proof, the Verifier generates random values $\gamma, r \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$, and sets $W := rP - \gamma X$. This results in a transcript (W, γ, r) which is valid by construction, and any transcript that resulted from a real interactive Schnorr proof can also be created using this method. Therefore, a transcript itself is not a proof that the Prover knows x – only the interactive protocol is. □

3.2.2 Notation

We will often use proofs of knowledge like Schnorr's. For briefness, we use notation that shows just the statement being proven. Using this notation, we denote the Schnorr proof of knowledge as

$$\mathrm{PK}\left[(x) : X = xP\right]$$

The notation implies that variables before the colon are only known to the prover, while all other variables mentioned are public information. Traditionally, the secret variables are named using the Greek alphabet. We will still do this if the secret is a compound expression; however, in cases like the above when the secret is just a variable, we prefer to use its existing name in the interest of clarity.

3.2.3 The Schnorr signature

Using a small modification, we can turn the Schnorr proof of knowledge into a signature. Instead of using it to say "I am the person who knows the discrete logarithm of X", we use it to say "The person who knows the discrete logarithm of X approves of message m."

This can be done using the following trick, which is called the *Fiat-Schamir heuristic*. Instead of requiring a Verifier to be present, we use a hash function to come up with the challenge, which takes as input not only the commitment but also the message. In effect, the protocol is no longer interactive.

Commitment First, the Signer (formerly Prover) generates a random number w in $\mathbb{Z}/q\mathbb{Z}$ and computes the nonce $W = wP$.

Challenge The Signer computes the challenge $\gamma = \mathcal{H}(m, W)$.

Response The Signer now calculates $r = \gamma x + w$. The resulting signature is the tuple (γ, r).

Verification On receipt of the message m and the signature (γ, r), the Verifier checks that $\gamma = \mathcal{H}(m, rP - \gamma X)$.

Note that the formula the Verifier uses in the hash function is the same as the one we used in the last step of Schnorr's proof of knowledge. Therefore, if the Signer made no mistakes, the hash value should be equal to γ.

3.2.4 Schnorr's blind signature

The Schnorr signature we saw has the property that everyone can see the message and the resulting signature. Sometimes, for example in an electronic voting scheme, this is not desired. In these cases we can use a blind signature scheme. This kind of schemes allows a Signer to provide a signature over a message to a Recipient, without seeing the final signature or the message. The Recipient can then show the message and its signature to a Verifier. This allows a voter to have her vote signed by an authority, who will sign only one vote for each voter. When counting the votes, the Verifier can not link a vote to its voter, even when colluding with the Signer.

With a few modifications, we can turn Schnorr's signature scheme into a blind signature scheme [Pointcheval and Stern, 1996]. The blind signature protocol is different from the regular Schnorr signature, because we do not want the Signer to learn either γ or r. Therefore the Recipient adds a random number to both of them, an operation we refer to as *blinding*. To make sure the signature still works, we also add these random numbers to the Signer's commitment. See table 3.2 for the step-by-step protocol.

Commitment First, the Signer generates a random number w in $\mathbb{Z}/q\mathbb{Z}$. She sends the nonce $W = wP$ to the Recipient.

Challenge The Recipient adds two nonce points, $\alpha X + \beta P$ to the Signer's commitment, which will later also be used to blind the challenge and the response. The blinded commitment is denoted $\widetilde{W}$. He then uses it with the hash function to create the challenge $\gamma = \mathcal{H}(m, \widetilde{W})$. He blinds the challenge by adding α and sends the result, $\widetilde{\gamma}$, to the Signer.

Response The Signer now calculates $r = \widetilde{\gamma}x + w$ and sends this value to the Recipient, who checks that it is correct and then blinds it by adding β. The result is denoted $\widetilde{r}$.

The final signature is $(\gamma, \widetilde{r})$.

Protocol 3.2: **Schnorr's blind signature**

Common information: group $\mathbb{E}$ generated by point P with order q, and a multiple X of P.
Private information for the Signer: the number $x \in \mathbb{Z}/q\mathbb{Z}^*$ such that $xP = X$.
Private information for the Recipient: the message m.

Signer		**Recipient**	**Comments**
select $w \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$			
send wP	$\longrightarrow$	into W	*Commitment*
		select $\alpha, \beta \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$	
		set $\widetilde{W} = W + \alpha X + \beta P$	
		set $\gamma = \mathcal{H}(m, \widetilde{W})$	
into $\widetilde{\gamma}$	$\longleftarrow$	send $\gamma + \alpha \mod q$	*Challenge*
send $\widetilde{\gamma}x + w$	$\longrightarrow$	into r	*Response*
		verify $W \stackrel{?}{=} rP - \widetilde{\gamma}X$	
		set $\widetilde{r} = r + \beta \mod q$	
The resulting signature is $(\gamma, \widetilde{r})$.			

To verify the signature, any Verifier can calculate $\mathcal{H}(m, \widetilde{r}P - \gamma X)$. The second input to the hash function is equal to

$$\begin{aligned}\widetilde{r}P - \gamma X &= rP - \widetilde{\gamma}xP + \beta P + \alpha X \\ &= wP + \beta P + \alpha X \\ &= W + \beta P + \alpha X = \widetilde{W},\end{aligned}$$

so the hash is indeed equal to γ.

4

Anonymous credentials based on U-Prove

The Schnorr proof of knowledge allows you to prove you know the discrete logarithm of X. Sometimes, it would be nice to be able to prove more refined statements.

For example, if you want to buy a beer, you need to prove that your age is at least 16. One possible way to use a Schnorr proof for this is to give everyone who turns 16 a secret key x_{16} for their birthday. However, this key contains no personal information, so they would probably give x_{16} to their underage friends as well, which makes the system useless. On the other hand, you could just bring your passport and let the retailer look you up in some database, but you wouldn't want him to know who you are if you buy beer every day and happen to be the mayor of the next town over.

This brings us to the U-Prove proof of knowledge, invented by Stefan Brands in 2000. We will discuss a simplified version.

4.1 Credentials

Suppose you have a document containing information about you, signed by some authority, not unlike a passport. It is usually possible to convert the information, and thus the document, into a number (or more generally, an element of some group). We then call each piece of information, like 'age', an *attribute*, and the collection of all attributes that belong to a user an

(a) All attributes of a Dutch passport. Easily tampered with in its bare form.

(b) The entire credential. The way it is printed acts as the government's 'signature'.

Figure 4.1: Illustration of the different parts of a credential

attribute commitment. An attribute commitment combined with a signature from a central authority is a *credential*.

In its simplest form, a credential is given to you by some authority, and you can show it to anyone who's interested. With a passport, this works fine, since the people you show it to can't easily copy it. On the other hand, if your credential is not a piece of paper but a digital string, anyone can remember it and use it too. This would make identity theft too easy, of course.

One way to make this more secure is to design the attribute commitment using a trapdoor function like point multiplication on an elliptic curve: given some points X_1, X_2 and some attribute values k_1 and k_2, the commitment is $C = k_1X_1 + k_2X_2$. This makes sure you can't infer the attribute values from the attribute commitment itself, but given the attribute values it is easy to build the commitment. Then, when you are showing it to someone, you also give a proof of knowledge to show that you actually know the attribute values that were used to build the commitment.

The U-Prove system combines this idea with the blind Schnorr signature. The advantage of using the latter is that the issuer never sees the recipient's credential, and therefore can't find out whom he shows it to, even if all alcohol salesmen collaborate with her.

We will next discuss the two protocols that together form the U-Prove system: one protocol that lets an authority issue a credential to a recipient, and one that lets the recipient show a credential to someone else.

4.2 Setup phase

Before anyone can use the U-Prove system, a couple of system parameters must be known to everyone. These should remain constant; if they didn't, nobody could check credentials (imagine if no two passports looked even slightly alike!). These parameters are generally chosen by an authority, whom we will call the Identity provider in the U-Prove context.

The Identity Provider chooses an elliptic curve group on $E_{a,b}(\mathbb{Z}/p\mathbb{Z})$ generated by a point P with order q. She decides how many attributes each credential should contain, this amount is called m. Finally, she picks $m+1$ secret nonzero random elements $y, \{x_i\}_{i=1}^m$ from $\mathbb{Z}/q\mathbb{Z}$. Her public key is then $Y = yP$. She also calculates $X_i = x_i P$ for each i.

The collection $(E_{a,b}(\mathbb{Z}/p\mathbb{Z}), q, P, Y, \{X_i\}_{i=1}^m)$ is the Identity Provider's public setting. She broadcasts it to everyone.

4.3 The issuing protocol

The protocol for issuing a credential takes place between a recipient, who in U-Prove is called the User, and the Identity Provider. These parties agree on the values of the m attributes $\{k_i\}_{i=1}^m$ used to construct the attribute commitment.

The issuing protocol itself is a blind signature scheme which looks like Schnorr's blind signature. The message being signed is the User's attribute commitment, C, as discussed in §4.1.

Random commitment First, the Identity Provider generates a random number w in $\mathbb{Z}/q\mathbb{Z}$. She sends the nonce $W = wP$ to the User.

Challenge The User adds $m+2$ nonce points, $\alpha(Y + \sum_{i=1}^m k_i X_i) + \beta P$ to the Identity Provider's commitment, denoting the result by $\widetilde{W}$. He then uses it with the hash function to create the challenge $\gamma = \mathcal{H}(C, \widetilde{W})$. He blinds the challenge by adding α and sends the result, $\widetilde{\gamma}$, to the Identity Provider.

Response The Identity Provider now calculates $r = \widetilde{\gamma}\,(y + \sum_{i=1}^m k_i x_i) + w$ and sends this value to the User, who checks that it is correct and then blinds it by adding β.

Protocol 4.1: **Issuing of a U-Prove credential**

Common information: Identity Provider's public setting, attribute values k_i, $i = 1, \ldots, m$ and $C_U = \sum_{i=1}^{m} k_i X_i$.
Private information for the Identity Provider: the number $y \in \mathbb{Z}/q\mathbb{Z}^*$ such that $yP = Y$ and the set $\{x_i\}_{i=1}^{m}$ such that $x_i P = X_i$ for each i.

Identity Provider		**User**	**Comments**
select $w \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$			
send wP	$\longrightarrow$	into W	*Commitment*
		select $k_0 \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$	*User's SK*
		select $\alpha, \beta \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$	*Blinds*
		set $C = k_0 P + C_U$	*User's AC*
		set $\widetilde{W} = \alpha(Y + C_U)$ $+ \beta P + W$	
		set $\gamma = \mathcal{H}(C, \widetilde{W})$	
into $\widetilde{\gamma}$	$\longleftarrow$	send $\gamma + \alpha \mod q$	*Challenge*
send $w + \widetilde{\gamma} y$ $+ \widetilde{\gamma} \sum_{i=1}^{m} k_i x_i$	$\longrightarrow$	into r	*Response*
		verify $W \stackrel{?}{=} rP$ $- \widetilde{\gamma}(Y + C_U)$	
		set $\widetilde{r} = r + \beta + \gamma k_0 \mod q$	

The resulting signature on the commitment C is $(\gamma, \widetilde{r})$.

4.4 The showing protocol

Once the user has acquired a credential, he can show it to someone. This is where the elaborate design of the attribute commitment plays an important role. Unlike the systems we discussed earlier, U-Prove makes it possible to show only some attributes, while keeping the others hidden. This makes sure the User does not have to give anyone more information than necessary.

Suppose the User has disclosed the values of the attributes with indices in the index set $\mathcal{D}$. He now wishes to prove that these values correspond with the ones in the attribute commitment C signed by the Identity Provider, while concealing all other attributes, which have indices in $\mathcal{C} = \{1, \ldots, m\} \setminus \mathcal{D}$. He can then engage in the following protocol with a Verifier.

Step 1 The User sends the credential to the Verifier, who checks that the signature is correct.

Step 2 The User and Verifier engage in a protocol much like Schnorr's Proof of Knowledge. Again, this consists of a *commitment* to all of the concealed attribute values, a *challenge* by the Verifier, and a *response* generated from the challenge, the attribute values and the nonces.

Protocol 4.2: **Showing of a U-Prove credential**

Common information: Identity Provider's public setting. A set of attribute values the User wants to disclose, $\{k_i\}_{i\in\mathcal{D}}$, the corresponding point in $\mathbb{E}$, $C_\mathcal{D} = \sum_{i\in\mathcal{D}} k_i X_i$, and the index sets $\mathcal{C}$ and $\mathcal{D}$.
Private information for the User: Concealed attribute values $\{k_i\}_{i\in\mathcal{C}}$, the attribute commitment C and the signature $(\gamma, \widetilde{r})$.

User		**Verifier**	**Comments**
		Step 1	
send $C, (\gamma, \widetilde{r})$	$\longrightarrow$	into $\overline{C}, (\overline{\gamma}, \overline{r})$	
		verify $\overline{\gamma} \stackrel{?}{=} \mathcal{H}(\overline{C}, \overline{r}P$	
		$\quad - \overline{\gamma}(\overline{C} + Y))$	*Verify sig.*
		Step 2	
select $w_i \in_\mathcal{R} \mathbb{Z}/q\mathbb{Z}$			
$\quad \forall i \in \{0\} \cup \mathcal{C}$			
send $w_0 P + \sum_{i\in\mathcal{C}} w_i X_i$	$\longrightarrow$	into W	*Commitment*
into γ	$\longleftarrow$	send $\gamma \in_\mathcal{R} \mathbb{Z}/q\mathbb{Z}^*$	*Challenge*
send $\gamma k_0 + w_0$,			
$\quad \gamma k_i + w_i \; \forall i \in \mathcal{C}$	$\longrightarrow$	into $r_0, \{r_i\}_{i\in\mathcal{C}}$	*Response*
		set $C_\mathcal{C} = \overline{C} - C_\mathcal{D}$	
		verify $W + \gamma C_\mathcal{C} \stackrel{?}{=}$	
		$\quad r_0 P + \sum_{i\in\mathcal{C}} r_i X_i$	*Verify POK*

We will now discuss the various security properties of the proof of knowledge in step 2, like we did with Schnorr's signature.

Proposition 1. *The U-Prove proof of knowledge is complete.*

Proof. At the end of the protocol, $C_\mathcal{C} = k_0 P + \sum_{i\in\mathcal{C}} k_i X_i$. Hence

$$\begin{aligned} W + \gamma C_\mathcal{C} &= (w_0 + \gamma k_0)P + \sum_{i\in\mathcal{C}} (w_i + \gamma k_i) X_i \\ &= r_0 P + \sum_{i\in\mathcal{C}} r_i X_i, \end{aligned}$$

so the verifier accepts. □

Proposition 2. *The U-Prove proof of knowledge is sound.*

Proof. The soundness of step 1 depends on the DL complexity assumption, and on whether the hash function is unpredictable enough to prevent the User from coming up with a pair $(\gamma, \widetilde{r})$ that successfully passes the verification. This is true by definition of the hash.

In step 2, if all attributes are disclosed, this proof is as sound as Schnorr's proof of knowledge. Otherwise, we can show soundness analogously to the proof of Schnorr's proof of knowledge, by rewinding the adversary after its response and giving it a different challenge. □

In contrast to Schnorr's proof, the U-Prove proof of knowledge is not unconditionally zero knowledge (see [Brands, 2000, §2.4.3] for a discussion). Instead, it has the following, slightly weaker property:

Witness-indistinguishability After executing two proofs of knowledge, a Verifier can not decide with confidence greater than 50% if the two provers had the same secret (the *witness*) or two different ones.

Proposition 3. *The U-Prove proof of knowledge is witness-indistinguishable.*

Proof [Brands, 2000]. We will show that for each proof of knowledge, as seen from the Verifier's side, the User could have used any $k_0, \{k_i\}_{i\in\mathcal{C}} \subset \mathbb{Z}/q\mathbb{Z}$ as his witnesses with equal probability.

Consider a specific transcript $W, \gamma, \{r_i\}$ seen by the Verifier. Suppose we suspect the User's witnesses were $\hat{k_0}, \{\hat{k_i}\}_{i\in\mathcal{C}}$; recall that they are still a DL-representation of a known and fixed part of his attribute commitment, $C_\mathcal{C}$. Since we know the User's responses r_i, we conclude that he must have chosen $\hat{w_i} = r_i - \gamma\hat{k_i}$. Since the Verifier does accept the responses, we see that

$$\begin{aligned}
\hat{W} &= \hat{w_0}P + \sum_{i\in\mathcal{C}} \hat{w_i}X_i \\
&= r_0P + \sum_{i\in\mathcal{C}} r_iX_i - \gamma\hat{k_0}P - \gamma\sum_{i\in\mathcal{C}} \hat{k_i}X_i \\
&= W + \gamma C_\mathcal{C} - \gamma(\hat{k_0}P + \sum_{i\in\mathcal{C}} \hat{k_i}X_i) \\
&= W + \gamma C_\mathcal{C} - \gamma C_\mathcal{C} = W.
\end{aligned}$$

In other words, any witnesses $\hat{k_0}, \{\hat{k_i}\}_{i\in\mathcal{C}}$ can certainly result in the given transcript. Since the nonces are chosen uniformly from $\mathbb{Z}/q\mathbb{Z}$, the possible witnesses are distributed uniformly as well. □

4.5 Combining the protocols

An important consideration which guided the design of U-Prove was the User's privacy. Since the Issuing protocol uses a blind signature scheme, the User's final attribute commitment is hidden from the Identity Provider. This

means that the Identity Provider and the Verifier can't find out whether or not an execution of the Issuing Protocol and one of the Showing Protocol belong to the same User (actually, the same secret key), even if they collude. We will define this formally:

Definition (Linkability)**.** Suppose we have an execution transcript for each of two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$. The two protocols are said to be *linkable* if an adversary exists who can, with more than 50% probability of correctness, tell if both transcripts were made by the same user.

A protocol can be linkable to itself, in which case the definition pertains to two executions of the same protocol.

Proposition 4. *The U-Prove Issuing and Showing protocols are unlinkable, if no attributes are disclosed.*

Remark. Without this assumption, the adversary could 'recognize' the witness used by the agent by the disclosed attribute values. If you disclose information that might identify you, you shouldn't be surprised when you are indeed identified; therefore, we consider only the security of concealed data.

Proof (sketch). The issuing protocol is a so-called blind issuing protocol. All information the issuer sees is blinded by adding a uniformly random value to it and is therefore, from the issuer's point of view, uniformly random. This makes it impossible to link an issuing transaction to anything, including transactions from the showing protocol. □

The Showing protocol is definitely linkable to itself; the User sends his public key and signature to the verifier, and he can't blind them without invalidating the signature. This means that a User who wants to be completely untraceable needs to destroy any credential after use, which in turn means he needs to request a fresh credential for every time he needs to present a valid credential.

Since storing many credentials (or getting a few credentials many times) can be problematic on limited systems like smart cards, this is a drawback of the vanilla U-Prove protocols. In the next chapter, we present a modification to the U-Prove system that allows the User to blind his credential each time he shows it, which makes all transactions unlinkable to one another.

5

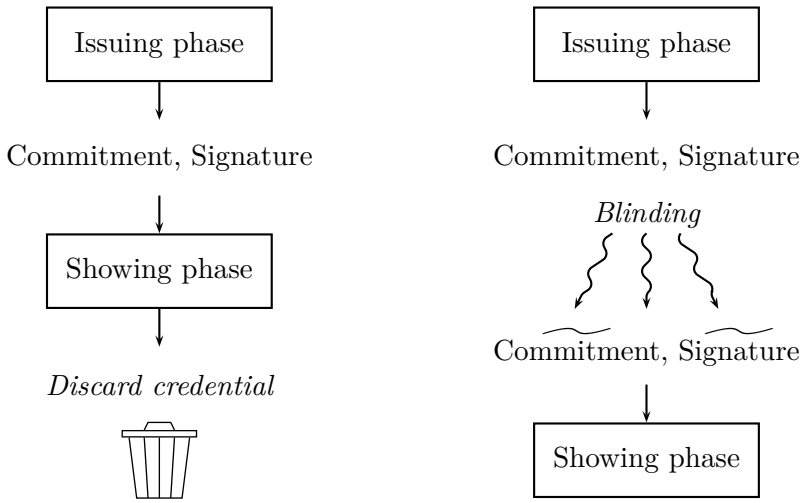
Extension to the U-Prove protocols

Using U-Prove, a User can obtain credentials from a central authority, the Identity Provider, and partially or completely disclose them to relying parties, the Verifiers, as we saw in chapter 3. The User's privacy is guaranteed as long as he does not recycle credentials that have been used, since the Issuing protocol is blinded, as in Schnorr's blind signature protocol. However, this requirement forces a privacy-conscious User to request and store many credentials.

We present a modified set of protocols that add a blinding operation to the Showing protocol. Using them, we gain unlinkability between different instances of the showing protocol, i.e. a malicious Verifier can not detect whether a given User has shown her his credential before. We retain untraceability of an issued credential, i.e. the Issuer and Verifier can't see whether or not two given instances of the Issuing and Showing protocol belong to the same user, even when colluding.

5.1 Design considerations

We would like a (secure) variation on U-Prove that allows us to modify a credential, such that different uses of the same credential are unlinkable to each other. This means that the User needs to be able to blind the entire credential.



(a) Vanilla U-Prove (b) Modified U-Prove

Figure 5.1: Illustration of our modification to the U-Prove system.

5.1.1 Structure of the credential

As in U-Prove, the User's credential consists of an attribute commitment

$$\text{Commitment} = \sum_i \text{User's } i^{\text{th}} \text{ attribute} \cdot i^{\text{th}} \text{ base point}$$

and a signature, which we will discuss later.

The base points $X_i = x_i P$ are public, but their discrete logarithms x_i are the Issuer's secrets. The first attribute, k_0, is the User's secret key, the other attributes k_i are known to both the User and the Issuer. To emphasize this distinction, we will denote the commitment by

$$C = k_0 X_0 + \sum_{i=1}^{m} k_i X_i.$$

Occasionally, we'll refer to the discrete logarithm of C as c. By construction, none of the participants knows c.

Note that the User's attributes have to be encoded as elements of $\mathbb{Z}/q\mathbb{Z}$. Several variations on this credential are possible, for example, the Issuer could encrypt some of the attributes so they can only be decrypted by the Verifyer. The User should be wary of this, since it is easy to store identifying information like a social security number this way.

5.1.2 Choice of the signature

A good way to achieve unlinkability is blinding, i.e. applying a transformation that changes the signature but retains its relationship to the attribute commitment. The traditional U-Prove signature is computed using a hash function. This makes blinding difficult. However, using a malleable signature introduces many risks. For example, a signature computed as

$$\text{Signature} = \text{Issuer's secret key} \cdot \text{User's attribute commitment}$$

can easily be verified using a pairing.[1] However, multiplying the commitment and the signature by a scalar will again produce a valid credential. The attribute values of the new credential are simply the old ones multiplied by the scalar. Hence, we would like a signature that remains valid under valid modifications (i.e., blinding) to the attribute commitment, but not under invalid ones (i.e. changes to meaningful attribute values).

These considerations have led us to use a signature scheme by Boneh and Boyen [2008]. Instead of the principle discussed above ($S = yC$), it is computed as

$$\text{Signature} = \frac{1}{\text{Issuer's SK} + \text{User's AC}} \cdot \text{Group generator}.$$

This type of signature can still be verified using a pairing, but it is no longer possible to easily forge signatures by multiplying the attribute commitment and the signature by a scalar. We can therefore safely blind the signature by multiplying it by a random scalar from $\mathbb{Z}/q\mathbb{Z}$.

Recall that we need to be able to blind the credential during the showing protocol to achieve unlinkability. Currently, the signature is only valid for C; using a small modification, we can make it valid to certain transformations of C as well. To do this we use two keypairs for the issuer, denoted by y, z for the secret keys and Y, Z for the public keys. For an attribute commitment $C = cP$, the signature consists of two values (S, r), with

$$S = \frac{1}{y + c + rz} Q \quad \text{and} \quad r \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}.$$

One can check the correctness of this signature by verifying that

$$e(Y + C + rZ, S) \stackrel{?}{=} e(P, Q). \tag{5.1}$$

Blinding the commitment is then done as follows. The User selects a random $\nu \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}^*$, and computes $\widetilde{C} = C + \nu Z$ and $\widetilde{r} = r - \nu$. Since

$$Y + \widetilde{C} + \widetilde{r}Z = Y + (C + \nu Z) + (r - \nu)Z = Y + C + rZ,$$

[1] Like this: $S = yC$ iff $e(S, P) = e(C, Y)$, since both are equal to $e(C, P)^y$.

we find that $(S, \widetilde{r})$ is a valid signature on $\widetilde{C}$.

This method allows us to randomize the attribute commitment without invalidating the signature. However, we are not done yet; at this moment, the User can still be tracked by the constant value $\widetilde{C} + \widetilde{r}Z$ or by the signature S itself. Hence, the signature values $(S, \widetilde{r})$ will have to be blinded separately. We will do this using a simple multiplicative blind.

Note that blinding the commitment this way allows us to treat the blind ν as an additional attribute, i.e. $\widetilde{C} + \widetilde{r}Z$ is constructed like a U-Prove attribute commitment. This allows us to use the existing U-Prove machinery during the showing protocol.

5.1.3 Issuing the signature

Computing a Boneh-Boyen signature seems at first to require that the Issuer knows c. This is not a nice solution, since the User needs to trust the Issuer completely; moreover, giving the Issuer the unblinded discrete logarithm of the signature will make it easy to link the Issuing and Showing protocols. Hence, we need a way for the Issuer and the User to jointly compute the sum

$$y + c + rz = y + k_0 x_0 + \sum_{i=1}^{m} k_i x_i + rz,$$

which allows the User to keep k_0 and r to himself, while the Issuer keeps y, z and the x_i-values to herself. As discussed, this sum still has the U-Prove commitment structure.

To meet this demand, we have chosen to use a form of homomorphic encryption invented by Paillier and Pointcheval [1999]. Denoting the encryption of a value x by $[\![x]\!]$, this system has the following two properties:

- $[\![x]\!] \cdot [\![y]\!]$ decrypts to $x + y \mod n$,
- $[\![x]\!]^y$ decrypts to $x \cdot y \mod n$.

These allow the User to encrypt his secret values, with which the Issuer can then perform a computation. The User can then decrypt the resulting ciphertext to obtain the result of the computation, with neither party gaining knowledge of the other's secret values. Both parties also blind the calculated values until the end of the computation. The protocol is discussed in detail in §5.4.

5.1.4 Efficiency

The system, and especially the Showing protocol, should be efficient enough for use with limited hardware, for example a smart card. The Verifier and

Issuer can be relied on to have reasonable computational power. This means that the User can not calculate pairings or perform computations with large numbers.

5.2 Stakeholders

Like U-Prove, we describe interactions between three kinds of agents: a **User**, who obtains and uses credentials, an **Issuer** or *Identity Provider*, who is charged with creating credentials for (and in cooperation with) Users, and a **Verifier** or *Relying Party*, who checks Users' credentials and relies on the correctness of the system (because, for example, he provides a service to authenticated users).

5.3 System setup

Most computations take place on an elliptic curve group $\mathbb{E}_1$ over $\mathbb{Z}/p\mathbb{Z}$, generated by a point P of prime order q, with additive identity $\mathcal{O}$. The size p of the underlying field should be at least 2^ℓ. A pairing e must exist on the curve; the second domain $\mathbb{E}_2$ of this pairing is an elliptic curve group generated by a point $Q\,(\notin \mathbb{E}_1)$. This should be a type III pairing, that is, there should be no efficiently computable group homomorphism $\mathbb{E}_2 \to \mathbb{E}_1$.

A (modified) U-Prove credential consists of an attribute commitment $C \in \mathbb{E}_1$ and a signature $(S, r) \in \mathbb{E}_2 \times \mathbb{Z}/q\mathbb{Z}$. The commitment is a discrete log representation of the user's secret key $k_0 \in \mathbb{Z}/q\mathbb{Z}$ and her attribute values $(k_i)_{i=1}^m \in \mathbb{Z}/q\mathbb{Z}^m$ with respect to the base points $(X_i)_{i=0}^m \in {\mathbb{E}_1}^m$:

$$C = k_0 X_0 + \sum_{i=1}^{m} k_i X_i.$$

The discrete logarithms $x_i \in \mathbb{Z}/q\mathbb{Z}$ of the base points $X_i = x_i P$ are known only to the Issuer and can be considered secret keys. For the Boneh-Boyen-like signature, the Issuer keeps two more keypairs, (y, Y) and (z, Z), where $Y = yP$ and $Z = zP$, both in $\mathbb{E}_1$.

The blind issuing of a Boneh-Boyen (BB) signature requires a Paillier-like cryptosystem, as discussed in section 2.2.2. The global modulus for this encryption scheme is denoted by n^2; $n = p_1 p_2$ with $q = p_1$.

5.4 Issuing protocol

To issue a blind (with repsect to the user's private key) BB-signature, we use a slight variation on [Hoepman and Lueks, 2012]. The issuer does not

need to know the entire commitment now, and therefore doesn't get it. He does get the attributes.

It should be noted that, due to the required size of n^2, it is not feasible to carry out the Issuing Protocol on a smart card. A trusted proxy should be used instead; for example, a User could receive the credential on his home computer or an issuing terminal, and then transfer it to a smart card.

5.4.1 Protocol description

Step 1 In order to blindly issue the signature, the user sets up a (possibly pre-generated) Paillier cryptosystem, and sends the Issuer the values of n and g, which he needs to encrypt values. The User keeps λ secret, for use during decryption.

Step 2 The Identity Provider and the User engage in a secure two-party computation protocol due to Hoepman and Lueks [2012], and the Identity Provider computes the signature 'inside' the Paillier encryption using its homomorphic properties. Both the User and the Identity Provider blind the values they send to protect the secrecy of their private keys.

Step 3 The User decrypts the computed, but still blinded signature value and the Identity Provider removes his blind.

The full protocol is shown in Protocol 5.1 on page 37.

5.4.2 Security

Proposition 1. *The signature issued by the Issuing Protocol is unforgeable.*

The issuing protocol relies on a Boneh-Boyen signature; strong existential forgery of this scheme is equivalent to the q-SDH-assumption. [Jao and Yoshida, 2009]

Proposition 2. *This Issuing Protocol is correct.*

Proof. We will demonstrate correctness by calculating the final verification step, assuming both parties have followed the protocol correctly. At the end

Protocol 5.1: **Issuing protocol**

Common input
System parameters $P \in \mathbb{E}_1, Q \in \mathbb{E}_2$ and Identity Provider's public data $Y, Z \in \mathbb{E}_1$
Attributes to be signed $(k_i)_{i=1}^m \in (\mathbb{Z}/q\mathbb{Z})^m$
Private input to the User
Secret key $k_0 \in \mathbb{Z}/q\mathbb{Z}$
Attribute commitment $C = k_0 X_0 + \sum_{i=1}^m k_i X_i \in \mathbb{E}_1$
Private input to the Issuer
Commitment basis $\{x_i\}$, secret keys $y, z \in \mathbb{Z}/q\mathbb{Z}$

User		**Issuer**	**Comments**
		Step 1	
set $(n, \lambda, g) =$ SetupPaillier			*Paill. setup*
send n, g	$\longrightarrow$	into n, g	
		Step 2	
select $\beta, r \in_{\mathcal{R}} [0, n)$			*Computing*
send $[\![\beta]\!], [\![\beta r]\!], [\![\beta k_0]\!]$	$\longrightarrow$	into $\overline{b}, \overline{r}, \overline{k}_0$	*the signature*
		select $\gamma \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}^*$	
		set $k = y + \sum_{i=1}^m k_i x_i$	
into $\overline{d}$	$\longleftarrow$	send $\overline{r}^z \cdot \overline{k}_0^{x_0}$	
		$\cdot\, \overline{b}^k \cdot [\![\gamma]\!] \mod n^2$	
Interactive PK to show that $\overline{d}$ was constructed as shown above			
		Step 3	
send $\mathrm{Dec}(\overline{d}) \mod n$	$\longrightarrow$	into $\overline{s}$	*Unblinding*
		set $s = \overline{s} - (\gamma \mod q) \mod q$	*the signature*
into $\widetilde{S}$	$\longleftarrow$	send $(s^{-1} \mod q)Q$	
set $S = \beta \widetilde{S}$			
Verify that (S, r) is a signature on C, as in (5.1)			

of step 2, the user receives

$$\begin{aligned}\overline{d} &= \overline{b}^{k}\overline{r}^{z}\overline{k}_0^{x_0}[\![\gamma]\!] \\ &= [\![\beta y + \beta\sum_{i=1}^{m} k_i x_i]\!][\![\beta r]\!]^{z}[\![\beta k_0]\!]^{x_0}[\![\gamma]\!] \\ &= [\![\beta(y + k_0 x_0 + \sum_{i=1}^{m} k_i x_i + rz) + \gamma]\!]\end{aligned}$$

The identity provider then removes γ in step 3, and sends

$$\widetilde{S} = \frac{1}{\beta(y + k_0 x_0 + \sum_{i=1}^{m} k_i x_i + rz)}Q$$

back to the user, who removes β by calculating $S = \beta\widetilde{S}$. Finally, the user verifies it in the usual way:

$$\begin{aligned}&e(Y + C + rZ, S) \\ &= e(yP + k_0 x_0 P + \sum_{i=1}^{m} k_i x_i P + rzP, \frac{1}{y + k_0 x_0 + \sum_{i=1}^{m} k_i x_i + rz}Q) \\ &= e(P, Q)\end{aligned}$$

□

We will now discuss witness-indistinguishability. This proof is broken up into two lemmas.

Lemma 3. *Step 2 of the protocol is witness-indistinguishable with respect to the User's secret key.*

Proof. Paillier encryption is semantically secure, which is equivalent to indistinguishability under a chosen-plaintext attack. Suppose we have an adversary who can, given two ciphertexts, decide whether or not they decrypt to the same plaintext. This adversary can be used to break the semantic security of the Paillier encryption.

In breaking it, we choose two random plaintexts m_1 and m_2, and send them to the encryption oracle, which replies with c for $i \in \{1, 2\}$. We encrypt m_1 to c_1, and ask our adversary whether c and c_1 originate from the same plaintext. If so, the oracle has encrypted m_1 for us; if not, m_2.

Hence, since the Paillier cryptosystem is semantically secure, different encryptions of the same secret key, i.e. the same witness, are indistinguishable from encryptions of different keys. □

Lemma 4. *Step 3 of the protocol is witness-indistinguishable with respect to the User's secret key.*

Proof. As calculated by the Identity Provider, $s = \beta(rz + k_0 x_0 + k)$. Note that the Identity Provider can't cheat by using fabricated values for its secret values, since it has to prove the correct construction of $\overline{d}$ before the User will decrypt it.

If r and β are chosen uniformly at random, the resulting sum s is also distributed uniformly at random, and therefore reveals no information about k_0. In particular, given two (possibly equal) secret keys, the resulting values of s are still distributed uniformly at random, so deciding whether or not the secret keys were equal will be a pure guess. □

Using these two lemmas and the fact that the interactive proof of knowledge in step 2 is a zero-knowledge proof of knowledge, we have

Proposition 5. *The Issuing Protocol is witness-indistinguishable with respect to the User's secret key.*

5.5 Showing protocol

The Showing Protocol allows the User to show some of the attributes in his credential to the Verifier, and prove that these attributes are legitimate using the Identity Provider's signature.

The showing protocol uses the original U-Prove selective disclosure protocol described in chapter 4, but precedes it with blinding operations that randomise the attribute commitment and the signature. If desired, the U-Prove showing protocol can be replaced by one of a set of protocols that prove various relations between attributes, see [Brands, 2000, Ch. 3].

To achieve unlinkability between executions of the showing protocol, we should refrain from sending S and r to the verifier. We can blind the signature by multiplying it with a random blind $\sigma \in \mathbb{Z}/q\mathbb{Z}^*$. This is not possible with r, since this will break the verification equation 5.1. We therefore slightly change the verification, and now check if

$$e(Y + C, S)e(Z, R) \stackrel{?}{=} e(P, Q), \quad \text{where} \quad R = rS.$$

Now, we have the User send blinded versions of $S, R := rS$ and Q to the Verifier, who then checks if the verification equation holds. To make sure the User doesn't cheat, he also uses Schnorr proofs of knowledge to show that he constructed his blinded signature correctly.

Our version of the showing protocol uses U-Prove's selective disclosure protocol nearly verbatim, adding only the extra attribute used to blind the commitment. However, any other U-Prove protocol can be substituted, as long as the additional blinding attribute is concealed.

5.5.1 Protocol description

Step 1 The User blinds the commitment and sends it to the Verifier.

Step 2 The User constructs and blinds the signature and sends it to the Verifier. The Verifier checks the signature.

Step 3 The User and Verifier execute a Schnorr proof of knowledge for each of the random values that blind the signature.

Step 4 The User proves knowledge of a DL-representation of the commitment with respect to P and $(X_i)_{i=0}^m$. This proof of knowledge closely resembles the U-Prove proof of knowledge by Brands, only adding the blind ν as a concealed attribute.

The protocol is listed in detail in Protocol 5.2 on page 41.

5.5.2 Correctness

The showing protocol is correct if an honest User can convince an honest Verifier that he has the credential he says he has. This means he must pass the verification in in step 2. Indeed, if the signature was calculated and blinded correctly,

$$\begin{aligned} e(P, \widetilde{Q}) &= e(P, Q)^{\sigma} \\ &= e(Y + C + rZ, S)^{\sigma} \\ &= e(Y + \widetilde{C} + \widetilde{r}Z, S)^{\sigma} \\ &= e(Y + \widetilde{C} + \widetilde{r}Z, \widetilde{S}) \\ &= e(Y + \widetilde{C}, \widetilde{S}) e(\widetilde{r}Z, \widetilde{S}) \\ &= e(Y + \widetilde{C}, \widetilde{S}) e(Z, \widetilde{R}); \end{aligned}$$

therefore, the showing protocol is correct.

Protocol 5.2: **Showing protocol**

Common input
System parameters and Identity Provider's public data
Index sets $\mathcal{C}, \mathcal{D}$
User's input from Issuing protocol
A signature (S, r) on attribute commitment C

User		**Verifier**	**Comments**
		Step 1	
select $\nu \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}^*$			*Blinding nonce*
set $\widetilde{C} = C + \nu Z$			
send $\widetilde{C}$	$\longrightarrow$	into $\overline{C}$	*Blinded commit*
send $\{k_i : i \in \mathcal{D}\}$	$\longrightarrow$	into $\{k_i : i \in \mathcal{D}\}$	*Selective discl*
		set $C_{\mathcal{D}} := \sum_{i \in \mathcal{D}} \overline{k_i} X_i$	
		Step 2	
select $\sigma \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}^*$			*Showing* $(S, \widetilde{r})$
set $\widetilde{S} = \sigma S$			
set $\widetilde{r} = r - \nu$			
set $R = \widetilde{r} S$			
set $\widetilde{R} = \widetilde{r} \widetilde{S}$			
set $\widetilde{Q} = \sigma Q$			
send $\widetilde{S}, \widetilde{R}, \widetilde{Q}$	$\longrightarrow$	into $\overline{S}, \overline{R}, \overline{Q}$	*Blinded sign.*
		verify $e(Y + \overline{C}, \overline{S}) e(Z, \overline{R})$	
		$\stackrel{?}{=} e(P, \overline{Q}) \wedge \overline{Q} \neq \mathcal{O}$	*Verify sign.*
		Step 3	
Interactive Schnorr PK$[\sigma : \widetilde{Q} = \sigma Q]$ and PK$[\widetilde{r} : \widetilde{R} = \widetilde{r} \widetilde{S}]$			
		Step 4	
select $k_0', \nu' \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$			*Start U-Prove*
select $k_i' \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}\ \forall i \in \mathcal{C}$			*pf of knowledge*
set $N = \nu' P + k_0' X_0$			
$\quad + \sum_{i \in \mathcal{C}} k_i' X_i$			
send N	$\longrightarrow$	into $\overline{N}$	*Commitment*
		select $\gamma \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$	
into $\overline{\gamma}$	$\longleftarrow$	send γ	*Challenge*
set $r_\nu = \overline{\gamma} \nu + \nu'$			
set $r_{k_0} = \overline{\gamma} k_0 + k_0'$			
set $r_i = \overline{\gamma} k_i + k_i'\ \forall i \in \mathcal{C}$			
send $r_{k_0}, r_\nu, (r_i)_{i \in \mathcal{C}}$	$\longrightarrow$	into $\overline{r_{k_0}}, \overline{r_\nu}, (\overline{r_i})_i$	*Response*
		verify $\overline{N} \stackrel{?}{=} \overline{r_\nu} P + \overline{r_{k_0}} X_0 +$	
		$\quad \sum_{i \in \mathcal{C}} \overline{r_i} X_i - \gamma(\overline{C} - C_{\mathcal{D}})$	*Verify POK*

5.5.3 Soundness

The Schnorr and U-Prove proofs are known to be sound. We still need to check step 2. If a dishonest User passes this verification, it has successfully forged a signature on his attribute commitment. We will use this forged signature to foil the Co-CDH complexity assumption:

Complexity Assumption (Co-Computational Diffie-Hellman (Co-CDH))**.** Suppose we have two groups, $\mathbb{E}_1$ and $\mathbb{E}_2$, with generators P and Q. Given aP, it is hard to calculate aQ. [Boneh et al., 2001]

Lemma 6. *Given an adversary who can forge a signature that passes verification in step 2 and 3 for a given C, one can break the Co-CDH assumption.*

Proof. For the Co-CDH problem, we are given $A = aP$ for some unknown a. We choose random c, y and z ourselves, calculate C, Y and Z, and set up an adversary to create a signature with these inputs. However, instead of P, we give the adversary A as the generator of $\mathbb{E}_1$.

The adversary supplies us with $\widetilde{C}, \widetilde{S}, \widetilde{R}$ and $\widetilde{Q}$. It also gives us a Schnorr proof of knowledge for σ, which we extract using the technique described in §3.2.1.

The verification equation in step 2 is

$$e(A, \sigma Q) = e(Y + \widetilde{C}, \widetilde{S})e(Z, \widetilde{R}),$$

which implies that

$$aQ = \frac{1}{\sigma}\left((y + c)\widetilde{S} + z\widetilde{R}\right).$$

This solves the Co-CDH problem. □

5.5.4 Linkability

Issuing to Showing protocol

For the purposes of this discussion, we will disregard the Paillier-encrypted values that the Issuer and the User exchange, since we assume that the Paillier encryption scheme is secure.

A run of the Issuing protocol yields the blinded logarithm of the signature $1/(\beta s)$, or alternatively $\hat{S} := \frac{1}{\beta}S$. Each run of the Showing protocol yields $\widetilde{C}_i, \widetilde{Q}_i, \widetilde{S}_i, \widetilde{R}_i$. We will show that given arbitrary values for each of these, there exist a corresponding r and a tuple $(\nu_i, \beta\sigma_i)$ for each run of the showing protocol that makes this run correspond to the Issuing protocol. Since these

exist for arbitrary transcripts, any run of the showing protocol has values for the blinds and the User's secret key that make it correspond to a given issuing transcript.

Proof. Select a random $r \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}^*$. For each showing transcript i, let $\widetilde{\sigma}_i := \log_{\hat{S}}(\widetilde{S}_i) = \beta\sigma_i$ and set $R'_i = 1/\widetilde{\sigma}_i \cdot \widetilde{R}_i$ (this is now equal to $\frac{1}{\beta}(r+\nu_i)S$). Then, set $\nu_i = \log_{\hat{S}}(R'_i) - r$ and $C_i = \widetilde{C}_i + \nu_i Z$. Now, the User could have used (C_i, S, r) as his private data, which would result in a protocol transcript

$$(C_i - \nu_i Z,\ \sigma_i Q,\ \sigma_i S,\ \sigma_i(r+\nu_i)S) = (\widetilde{C}_i, \widetilde{Q}_i, \widetilde{S}_i, \widetilde{R}_i). \qquad \square$$

In particular, if there is only one transcript from the showing protocol, there is an r for every $C \in \mathbb{E}_1$; hence, even if a user is forced to forfeit his secret key, it is impossible to decide whether or not a single run of the showing protocol was performed by him.

Between executions of the Showing protocol

It is known that the U-Prove proof of knowledge in step 4 is witness-indistinguishable [Brands, 2000]. Since we use a different attribute commitment each time, the linkability problem in the original U-Prove protocol is gone; if ν is chosen universally at random, the resulting $\widetilde{C}$ is distributed universally at random in $\mathbb{E}_1$ as well.

While we conjecture that the blinding of the credential and signature is secure, we have been unable to prove this using any standard complexity assumptions. Obvious candidates are assumptions about pairings, since the additive blinds (ν) occur in both elliptic curve groups. These fall into two categories:

1. Computational assumptions: "given some elements of $\mathbb{E}_1$ and $\mathbb{E}_2$, it is hard to compute a particular element of the pairing's image;"
2. Decisional assumptions: "given some elements of the elliptic curve groups $\mathbb{E}_i$ and an element t in the pairing's image, it is hard to decide whether t is equal to a particular element of the pairing's image."

If we were to attempt to use such assumptions, the adversary we use to solve the complex problem would have to be able to link two showing phases, which is a decisional task. Hence, we would like to use a decisional assumption. We have to feed the inputs mentioned in the assumption to our adversary, and our adversary takes as its input two transactions from the Showing protocol. The problem arises here: how are we supposed to feed t to the adversary, if the Showing protocol does not result in any elements from the pairing's image?

Another category of assumptions is those about product groups. These are groups $\mathbb{E}_1$, $\mathbb{E}_2$ of equal, known prime order q – the kind typically used as the domain of a pairing. However, the hard problems don't explicitly use any pairings. A particularly promising assumption from this category is the following:

Complexity Assumption (Decision Linear Problem version 2 (D-Linear2))**.** Given $P \in \mathbb{E}_1, aP, bP, acP, bdP$ and $Q \in \mathbb{E}_2, aQ, bQ$ and $T \in \mathbb{E}_1$, it is hard to decide whether $T = (c + d)P$. [Boyen and Waters, 2006]

Since we use the type-3 variety of this assumption (i.e. there is no easily computable isomorphism $\mathbb{E}_1 \to \mathbb{E}_2$), we can switch the two groups used in the assumption. This would seem to be a logical decision since the Showing protocol offers only one point in $\mathbb{E}_1$ and three in $\mathbb{E}_2$ for each run. However, we have been unsuccessful in trying to reduce the linkability problem to the problem posed in D-Linear2.

5.5.5 Efficiency for the prover

Let $c = |\mathcal{C}|$. Then the prover performs

$c + 2$ point additions,

$c + 9$ point multiplications,

$c + 5$ field additions and

$c + 4$ field multiplications.

5.5.6 Summary of security requirements

We are assuming the hardness in general of DL and DLREP, DDH and CDH. The Paillier system further uses the decisional composite residuosity assumption (DCRA).

The security parameter ℓ, in bits, defines the key size used in the main scheme. In general, larger is more secure but slower. For elliptic curve cryptography, a typical choice is $\ell = 128$.

Since the fastest algorithms for DL on a curve (for example Pollard's rho) take $\mathcal{O}(\sqrt{n})$ steps, the size of the underlying field should be a prime of size roughtly 2ℓ, i.e. $p \approx 2^{2\ell}$. Hasse's theorem states that the number of points in the elliptic curve group is close to p. For ECC, this number of points q should also be prime.

The pairing introduces some requirements: the main curve's embedding degree should be small, and there should be no efficiently computable homomorphism from the second elliptic curve group to the first. Furthermore, the discrete logarithm problem should be hard in the pairing's co-domain.

6

Discussion

In this thesis, we have discussed the U-Prove system by Brands for issuing and disclosing an electronic credential. In particular, we have treated a disadvantage of the system: to maintain full unlinkability of transactions, the user should not use a credential more than once. This means users have to obtain many credentials, which can be impractical if they have to be stored on limited devices like smart cards.

In an effort to solve this problem, we have proposed a new set of protocols for issuing and disclosing a credential. Though the new method is not compatible with 'old' U-Prove credentials, it does allow for the same kind of attributes, and the existing U-Prove protocols based on the DLREP function can be rewritten to work with the new credentials.

We have proven correctness and soundness for these protocols, and unlinkability between the protocols for issuing and showing credentials. While we conjecture that different executions of the showing protocol are also unlinkable, we have not been able to prove this. The main cause for this is the fact that we use both additive and multiplicative blinds during the showing protocol. Few computational complexity assumptions are known for product groups on which a pairing exists, that accomodate both types of blinding at the same time.

6.1 Future work

A few options remain for assessing the unlinkability of the showing protocol. An obvious approach, if difficult, is to find or define more complexity

assumptions that are applicable to the situation. This most likely includes assumptions about product groups, i.e. the product of two groups of known (and equal) prime order; while assumptions about pairings might be feasible, a situation like ours does not seem to call for them.

One could also avoid reductionist proofs entirely in favour of other techniques we have not attempted. One such technique is the ideal functionality model [Canetti, 2000] for protocol security. This model proposes a trusted third party which receives the secret inputs for all parties, and then computes the output of the protocol for each party and sends it back to them. An adversary controls one of the participants, and the protocol is considered secure if the adversary learns no more from it than from an interaction with the trusted third party.

Bibliography

Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21: 149–177, 2008.

Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Proceedings of Asiacrypt 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–32. Springer, December 2001.

Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 290–307. Springer, 2006.

Stefan A. Brands. *Rethinking public key infrastructures and digital certificates: building in privacy.* MIT Press, 2000.

Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000.

Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985.

Jaap-Henk Hoepman and Wouter Lueks. Towards efficiently revocable self-blindable credentials. Unpublished manuscript, 2012.

David Jao and Kayo Yoshida. Boneh-boyen signatures and the strong diffie-hellman problem. In Hovav Shacham and Brent Waters, editors, *Pairing-Based Cryptography – Pairing 2009*, volume 5671 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2009. URL `eprint.iacr.org/2009/221.pdf`.

Moni Naor, Yael Naor, and Omer Reingold. Applied kid cryptography. 1999. URL `www.wisdom.weizmann.ac.il/~naor/PAPERS/waldo.ps`.

Pascal Paillier and David Pointcheval. Efficient public-key cryptosystems provably secure against active adversaries. In *Advances in Cryptology*, ASIACRYPT '99, pages 165–179, London, UK, 1999. Springer.

David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology âĂŤ ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 1996.

Joseph H. Silverman and John T. Tate. *Rational Points on Elliptic Curves*. Springer, 1992.

Nigel P. Smart. *Cryptography: an introduction*. McGraw-Hill education. McGraw-Hill, 2003.

Latanya Sweeney. Comments to the Department of Health and Human Services on standards of privacy of individually identifiable health information. April 2002. URL `web.archive.org/web/20040324182628/http://privacy.cs.cmu.edu/dataprivacy/HIPAA/HIPAAcomments.html`.