University of Groningen

Master's Thesis

Implementing U-prove extension protocols in Magma

Author:

Marjolein van Linschoten

Supervisors:

Prof. dr. J. Top

Prof. dr. E. C. Wit

January 2014

university of groningen

faculty of mathematics and natural sciences

"How long do you want these messages to remain secret? ... I want them to remain secret for as long as men are capable of evil."

Neal Stephenson

Abstract

Implementing pairings seems easier than it actually is. The aim of this research was to implement the extension protocols of U-prove, designed by Erik Weitenberg, which contain a pairing. The ambition was to find out how difficult it is to implement a pairing. Another reason was to determine whether the extension protocols are applicable and not only work on paper. The implementations are executed in Magma. To understand how Magma works basic protocols are implemented. The theory needed for the implementations is explained. Both extension protocols are successfully implemented with the suitable example that is found. The obstacles that need to be overcome, for this single example, show the difficulty of implementing a pairing.

Acknowledgements

Primarily, I would like to thank my parents for their ceaseless support during my studies. Furthermore, special thanks to Mr. Top for sharing his knowledge and time with me. Also, I thank Mr. Wit for his effort and patience. Finally, I would like to express my gratitude to Erik Weitenberg for his help during the process.

Contents

List of Protocols

To my mathematics friend

Chapter 1

What to expect

When I started studying mathematics, I did not know in what direction I wanted to go. There was no specialization that I liked in particular. During some Master courses I got acquainted with cryptography. The puzzles behind cryptographic systems fascinate me over and over again. This is why my Master's thesis could not be about a topic outside the field of cryptography. One year, of my two-year Master's, I will study at the Science and Society department therefore this thesis encompasses 31 ECTS.

The goal is to implement the extension protocols of a system called U-prove in Magma. The principle of U-prove will be explained extensively in this thesis before the implementation is set out. Magma is an online software package, though it can also be used offline when a license is bought. The environment is specifically designed for algebraic computations. For example, when one takes two points on an elliptic curve, adding these two points is properly done according to the addition and doubling group law of elliptic curves. I used the online version [Magma]. This has one (dis)advantage: the maximal computation time is 120 seconds. This is a disadvantage, because this is a very strict limit to what one can compute. After 120 seconds it breaks off the computation. However it can also be seen as an advantage. This has everything to do with the purpose of U-prove.

U-prove involves proving you are the person you say you are. This is done by data saved on a smart card. A smart card is a pocket-sized card, like your debit card, and therefore has a small storage capacity. The fact that Magma breaks off the computation is thus a advantage, because the computations must not take too long. When a program takes longer than 120 seconds to compute, this is a sign it is not very suitable for smart card use. However, Magma is not a program used in practice. Though the step from paper to Magma is much bigger than the step from Magma to an environment that is used in practice.

The focus of this thesis lies on implementing cryptographic protocols in Magma. Deep theory behind these protocols does not lie in the scope of this thesis. Nevertheless, a result of this thesis is the realization that one is not able to implement the protocols without the use of algebraic theory. The actual implementations can be found as appendices.

This thesis is perfectly readable for someone with some algebraic knowledge. For people without knowledge about groups, rings, fields, etcetera it is probably more difficult material, though the concept of a protocol and the advantage of the extended protocols over the standard U-prove should be understandable. However, the mathematical content will be hard to grasp, but this can also be seen as a challenge.

The content is divided into chapters. The following chapter states the inducement of this research. Chapter 3 sets out some cryptographic aspects like credentials and U-prove. Furthermore, some background information on elliptic curve cryptography is given. To grasp the concept of a protocol some well known protocols are implemented in Chapter 4. In Chapter 5 the standard U-prove protocols are introduced. The extended protocols, the concept of a pairing and an approach to implement these extended protocols are extensively set out in Chapter 6. Finally, a conclusion will be drawn.

Chapter 2

Inducement

2.1 First motive

Last year a Master's student in Mathematics, Erik Weitenberg, wrote his thesis at the University of Groningen. I know him but I was not aware of the fact that his field of interest is cryptography. He did an internship at 'Nederlandse Organisatie voor Toegepast Natuurwetenschappelijk Onderzoek' (TNO) also in Groningen. This resulted in an interesting thesis, which I was glad to read. The security protocols Weitenberg worked on look usable on paper, although the question remained whether they are applicable. How easy or difficult is it to apply them in practice? This was one motive to start this research. The reader will, after reading Weitenberg's thesis, see some similarities between the building of his and this thesis, which is not strange since I tried to implement the protocols he worked on. Furthermore, some of the concepts I explain can also be found in Weitenberg's thesis, [Weitenberg, 2012], but it is necessary to mention them in order to follow this thesis. Of course this thesis also includes new concepts to understand the implementations.

2.2 Second motive

The most important concept I needed to explore to implement the extension protocols of Weitenberg is the *pairing*. It is a remarkable issue that most of the security protocols that are designed are treating pairings like a 'black box'. This is not necessarily a bad approach, since the selection and implementation of a pairing can be quite complex. It allows one to ignore the mathematical and algorithmic difficulties and focus entirely on the cryptographic part of the research. The designers are only making use of the properties of pairings (which will be discussed in chapter 6), but that is far from sufficient

to see if a pairing is suitable. There are a lot of researchers aware of this, but there are many more who are not. The result is that the authors easily make wrong assumptions concerning the properties of pairings. The protocol may not be as applicable in practice as assumed or may not be as efficient as assumed by the authors. [Galbraith et al., 2006] This is also what happens in Weitenberg's thesis. The concept of pairings is shortly mentioned, but not really elucidated. When it comes to implementing these protocols, you can not go around this. You need an explicit pairing which suits the protocol. This was the second motive for this research.

Chapter 3

Several cryptographic aspects

The shortest way to explain what cryptography is, is the following: the study of secure communication between two parties in the presence of a third party called the *adversary*. For example, one can think of two people e-mailing each other, of course someone else besides those two is not allowed to read those e-mails. A money transaction between a bank and his client is another example.

When one finds it hard to understand how this works in practice, imagine two people Alice and Bob. Alice wants to tell Bob a big secret, but all communication can only go through a public channel. A public channel means that every eavesdropper (adversary) can see the information Alice and Bob send to each other. Do not let it be a surprise the eavesdropper is called Eve.

The scheme that should solve this problem is called a *protocol*. A protocol contains the steps that Alice and Bob need to perform in order to send each other a message. A protocol is to be read from top to bottom. Some protocols are safer than others. The security of protocols is based on *complexity assumptions* that are, as the name predicts, assumed to be hard to break. Said in another way: when you break the assumption, you can break the security of the protocol.

In this chapter three aspects of cryptography, that are needed in the next chapters, are explained in the first section; credentials, U-prove and the hash function. Elliptic curves have a special addition group law, which will be explain in the second section. After that there is some explanation why one must look at these curves over finite field and not just finite rings. In the last subsection some known attacks to break a system that uses elliptic curves will be briefly set out.

3.1 Needed aspects

Now the basic concept of cryptography is clear one needs to understand the following aspects.

3.1.1 Credentials

The liquor-store-model is a model that is widely used to explain credentials; Weitenberg [2012] also used this model. Assume one goes to the liquor store to buy something you definitely need to be at least 18 years old for. Than the friendly woman behind the counter will ask for a proof that this is indeed the case. Of course one needs something really convincing for her. A driving license or identification card are more suitable than a client card from a certain store. Why? Because the first two contain your date of birth and are provided by the government. Cards from the government are hard to falsify and trustworthy. A client card is not provided by the government and therefore not trustworthy. The reason for this is the possession of special features of those trustworthy cards; like the perforation that forms the picture of the owner. These special features, that are added by the government to make the card legitimate, are called the *signature of the Issuer*; is this case the Issuer is the government. The woman behind the counter is the *Verifier*.

Such properties, like 'age', can usually be converted into a number (or more generally, an element of a group). After the convertion this property is called an *attribute* of the *User*. In the liquor-store-model the User is the one who wants to buy the liquor. The collection of all the attributes that belong to the User is the *attribute commitment*. The attribute commitment from the User (the statement), together with the signature from the Issuer (the way to verify the statement) is a *credential*. It can be given to you by the government, like a passport. A passport can hardly be remembered if someone sees it for a few seconds; it is hard to copy. A credential can be your identification card, driving license or the combination of the username and password of your e-mailaccount. [Weitenberg, 2012]

When it comes to a smart card, where the credential is a string of bits, it is not hard to remember this credential and identity theft is easily accomplished. To give a small taste of what will be discussed in chapter 5 look at the way one can go around this problem. For example, there are two attributes with two attribute values, k_1 and k_2. One takes two points on an elliptic curve, X_1 and X_2. Now the attribute commitment is $C = k_1X_1 + k_2X_2$. It is easy to build the commitment when the attribute values are known, but one cannot easily find the attribute values from the commitment. [Weitenberg, 2012]

3.1.2 U-prove

Weitenberg's thesis concerns showing credentials to others via smart cards. In this case, as discussed above, the User should not want to give his/her credential to the Verifier. The Verifier can than copy the credential and pretend to be the User. Looking back at the aim, described in the previous subsection, it is not necessary to show the credential. It is sufficient when the User proves to have the right credential and that he/she is the person the Issuer assigned it to. The extension protocols Weitenberg constructed are an extension of something called *U-Prove*. U-Prove is a solution to this problem. It was designed by Stefan A. Brands in 2000 and shows a way to reveal credentials partially and to prove that a credential consists of certain information, for example that the User is older than 18. With U-Prove the woman behind the counter will not know if the User bought liquor yesterday; even if she is a secret agent who knows all the credentials ever assigned. [Weitenberg, 2012]

However, there is a downside to this: once the User used a credential it becomes useless. When the User uses it again this can be recognized so the User needs a new credential after one is used to maintain his/her privacy. Unfortunately, a smart card has its limitations when it comes to memory storage and constantly asking for a new credential is not how people like to spend there days. [Weitenberg, 2012] Finding a way around this is the extension Weitenberg worked on.

3.1.3 Hash function

Some of the protocols that will be discussed use the concept of a *hash function* $\mathcal{H}$. A hash function is an algorithm that maps elements to elements in a certain set. It is a deterministic algorithm, this means a particular element will always have the same output element. When a hash function is deterministic, the security of this function depends on the difficulty of three aspects. The security depends on the hardness of: finding an m when $\mathcal{H}(m)$ is given, finding m_1 and m_2 such that $\mathcal{H}(m_1) = \mathcal{H}(m_2)$ and finding an m_2 given m_1 and $\mathcal{H}(m_1)$ such that $\mathcal{H}(m_1) = \mathcal{H}(m_2)$. Furthermore, a good hash function is uniformly distributed. This means, all the output elements have the same probability of being the hash value of a certain input element. This increases the security of the hash function. When one takes the hash function of an element a, one says a is *hashed*.

3.2 Elliptic curves in cryptography

In this thesis the focus will be on curves without singularities of the *Weierstrass form*

$$y^2 = x^3 + ax + b.$$

These are *elliptic curves* with no cusps or nodes. To apply them one must look at these curves over a finite field.

3.2.1 Addition and doubling group law

The elliptic curves that are studied are defined over a finite field K, such that K does not have characteristic 2 or 3. The coefficients a and b of the curve are elements of K. The curve will, for example, look like the curve in figure 3.1. The points on the elliptic curve over K form a group. When E is the elliptic curve, the elliptic curve group over K is denoted $E(K)$. The elements of the group, $(x, y) \in K \times K$, are the points for which the equation of the curve holds, together with the point at infinity $\mathcal{O}$. The point at infinity is the 'zero' of the group. Furthermore, when $P = (x_p, y_p)$ the additive inverse is $-P = (x_p, -y_p)$.

The group is defined such that when three points lie on one line, their sum is $\mathcal{O}$. The point at infinity lies, thus, on every vertical line. This makes it possible to define an additive group law for this group, noted as $\bigoplus$. To add two points P and Q one must draw a line between those two points and find the third intersection point R of the curve and the line. Then one must reflect R with respect to the x-axis. This will give the additive inverse $-R$; this point is equal to $P \bigoplus Q$. (And with that R equals $-(P \bigoplus Q)$.) When one wants to add a point P to itself, doubling, the tangent line at P needs to be drawn. To make sure the correct additive group law is used, $P \bigoplus P$ is written as $[2]P$ and not as $2P$. The addition and doubling of points is shown in figure 3.1.

Realize that the curve in figure 3.1 is the representation of a curve over $\mathbb{R}$ and not over a finite field. To understand when a point lies on a curve over a finite field look at the following example.

Example. Consider the elliptic curve $y^2 = x^3 - 3x + 3$ over $\mathbb{F}_7$. There does not exist a point with x-coordinate equal to 0, because there is no y in $\mathbb{F}_7$ such that $y^2 = 3$. A point with x-coordinate equal to 1 does exist. Than $y^2 = 1$ and this hold when y equals 1 or -1; the last one equals 6 modulo 7. This method can be continued for all possible x-coordinates in $\mathbb{F}_7$. The group elements on this elliptic curve over $\mathbb{F}_7$ are:

(1,1), (1,6), (2,3), (2,4), (3,0), (5,1), (5,6), (6,1), (6,6) and $\mathcal{O}$.

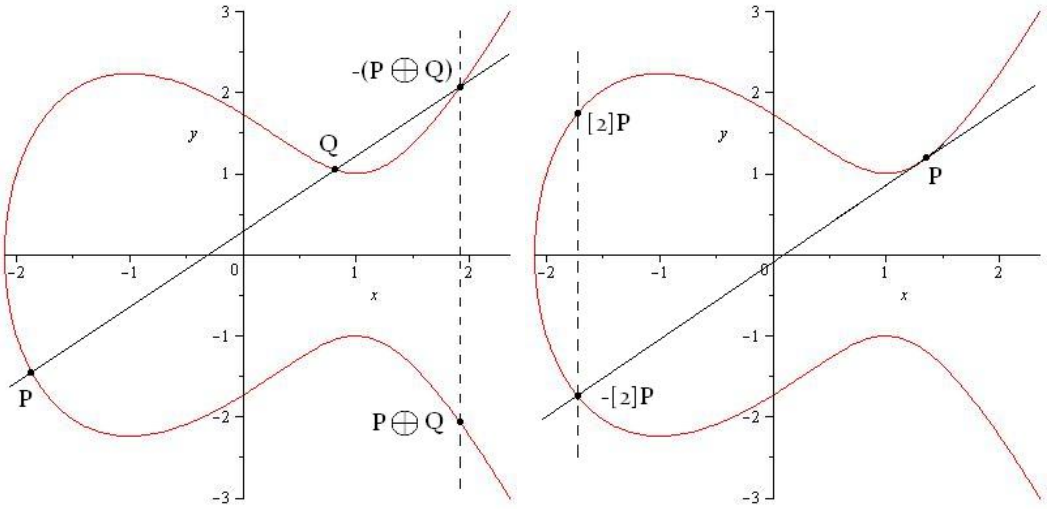

FIGURE 3.1: Addition and doubling of points on $y^2 = x^3 - 3x + 3$.

Magma displays a point of an elliptic curve group in three coordinates. A point (α, β) is denoted as $(\alpha : \beta : 1)$. The element at infinity is displayed as (0:1:0). This must be kept in mind when one wants to implement in Magma.

3.2.2 Why elliptic curves over finite *fields*?

Curves can be defined over all sorts of sets and rings. Why is it wise to look at elliptic curves over finite fields and not finite rings when it comes to cryptography? In most of the literature the author assumes it is an obvious thing to do.

Now let us come back to the complexity assumptions mentioned in the beginning of this chapter. Complexity assumptions are always assumed to be hard to solve, i.e. nobody has yet found an efficient way to state that these problems are always easy to solve. There are two complexity assumptions that will be used to answer the main question of this subsection.

Complexity assumption *Elliptic Curve Discrete Logaritm Problem (ECDLP).* Given an integer n and $E(\mathbb{Z}/n\mathbb{Z})$, an elliptic curve over the ring $\mathbb{Z}/n\mathbb{Z}$, and let $P, Q \in E(\mathbb{Z}/n\mathbb{Z})$ be points on this curve. It is hard to find the integer m such that $Q = [m]P$.

When one wants to compute $P+Q$, a fraction needs to be computed. This is not possible when the denominator is not a unit. There is a solution to go around this, but this does not lie in the scope of this thesis.

Complexity assumption *Factoring n.* Given an integer n, it is hard to find the prime factorization of n.

All these complexity assumptions hold for very large n. For example it is not hard to find the prime factorization of 21 (= 3·7). And for small n one can just tries every possible m to find the right one, which is not hard to do. But what is 'hard'? Problems are said to be *hard* to solve when there does not exist an algorithm that solves the problem in polynomial time. A problem of size N is solvable in *polynomial time* when there exists an algorithm that needs l steps to be completed; where l is a polynomial function of N.

When one defines an elliptic curve over $\mathbb{Z}/n\mathbb{Z}$, where n is a composite squarefree integer, counting the number of points of this curve is randomly polynomial time equivalent to factoring n. When one looks to the general case, the elliptic curve is defined over an arbitrary ring R and it can be decomposed into finite local rings; $R = \bigoplus_{i=1}^{n} R_i$. If the reduction of $R \to R_i/m_i$, with m_i the maximal ideal of the local ring R_i, can be computed efficiently, then the number of points of the elliptic curve over R can be counted in deterministic polynomial time using a well known algorithm. [Fontein, 2005]

Also when one considers an elliptic curve group of a curve over $\mathbb{Z}/n\mathbb{Z}$, with n a composite square free integer, something interesting happens. If the ECDLP in this group can be solved, then n can be factored in randomly polynomial time. Furthermore, if one is able to factor n, one can reduce the ECDLP of this curve E to ECDLPs in $E(\mathbb{Z}/p\mathbb{Z})$ for all prime factors p of n in deterministic polynomial time. Again, for the general case, if the reduction $R \to R_i/m_i$ can be computed efficiently, then the ECDLP of the elliptic curve over R can be reduced to the ECDLP of the same curve over R_i. The m_i is the maximal ideal of the local ring R_i. [Fontein, 2005]

The reason elliptic curves are used in cryptography is because with relatively small fields one can get a good security level, i.e. the ECDLP is hard enough. When one uses a finite ring R instead of the finite field $\mathbb{F}_q$, with R about the same cardinality as $\mathbb{F}_q$ and R can be effectively decomposed, then the ECDLP and the point counting problem for elliptic curves over R are much easier than for elliptic curves over $\mathbb{F}_q$. To avoid this, one has two options: R must be a ring whose components R_i are large enough such that the discrete logarithm problem over these components is still hard or R must be a ring which cannot be decomposed. The first is not a good option, because adding two points on an elliptic curve is much more difficult than one multiplication in R and the field over which the elliptic curve is defined can be relatively small. For the second option these are exactly the local rings and therefore is must be hard to effectively compute R/m, where m is the maximal ideal of R. It appears to be relatively easy to compute R/m. [Fontein, 2005]

Therefore it is not useful to use elliptic curves over finite rings in cryptography. If the reader is interested in more detailed information about this topic, I would like to refer to the source I consulted for this subsection.

3.2.3 Known attacks

There are many researchers who try to break the security of existing protocols. They try to find the weaknesses of the protocols. The way to find flaws is finding algorithms that break the complexity assumption the protocol is based on. It can happen that a protocol has a flaw when an element of the protocol possesses a certain property. This property should be avoided to keep the protocol secure. Evidently one should avoid this property or the complete protocol. In this subsection known attacks for the complexity assumptions mentioned in the previous subsection will not be explained, but a conclusion will be drawn from them.

Let E be an elliptic curve over a finite field $\mathbb{F}_q$ and let N be the order of the group $E(\mathbb{F}_q)$. Take a point $P \in E(\mathbb{F}_q)$ of order r.

Elliptic Curve Discrete Logaritm Problem (ECDLP)
Remember, the problem is the hardness of finding m, given $[m]P$ and P.

- *Pohlig-Hellman (1978)* This algorithm shows that the time it takes to solve the ECDLP depends only on the largest prime p dividing r, because of the Chinese Remainder Theorem [Silverman, 2009]. The running time is approximately $O(\sqrt{p})$. Therefore r must be divisible by a large prime p. This is one of the reasons why it is logical to choose P such that r is a prime of at least 160 bits. The same conclusion can be drawn from Pollard's ρ algorithm. [Tezcan, 2011]

- *Menezes-Okamoto-Vanstone (MOV) (1993)* This attack reduces the ECDLP of $E(\mathbb{F}_q)$ to the easier discrete logarithm problem in $\mathbb{F}^*_{q^k}$; which is a similar problem but this problem contains integers instead of points on elliptic curves. When $gcd(r, q) = 1$ one can take k to be the smallest integer such that $q^k \equiv 1(\mathrm{mod} r)$. This reduction can be done in polynomial time, with respect to the number of operations in $\mathbb{F}_{q^k}$. To avoid this, $q^k - 1$ should not by divisible by r for each $1 \leq k \leq C$ ($C = 20$ is sufficient). When E is supersingular this cannot be achieved (see 6.3 for the definition of a supersingular curve). The conclusion is: do not work with supersingular elliptic curves. [Tezcan, 2011]

- *Semaev (1998), Satoh-Araki (1998), Smart (1999)* When $N = q$ the system will resist the MOV attack, but this method solves the ECDLP. The conclusion is: $N \neq q$ should hold. [Tezcan, 2011]

Factoring n
Do not forget, the problem is the hardness of finding the prime factorization of n.

- *Pollard's $p-1$ (1974)* Pollard's $p-1$ algorithm is capable of factoring large numbers, but only numbers that satisfy a certain condition. The algorithm is a valuable factorization tool, though it is only applicable to numbers with a prime factor p such that $p-1$ is smooth, hence the name of the algorithm. A number is said to be smooth when it is the product of small primes. [Silverman, 2009] To avoid factorization of large numbers, one must make sure the numbers do not have a prime factor p such that $p-1$ is smooth.

- *Lenstra's elliptic curve factorization (1987)* Lenstra used elliptic curves to factorize a large number N. The crux of this algorithm lies in finding an elliptic curve such that for some prime factor p of N, the order of the group $E(\mathbb{F}_p)$ is a smooth number. [Silverman, 2009] The worst scenario for this algorithm is when the second largest prime divisor of N is not much smaller than $\sqrt{N}$, so that N is the product of some small primes and two large primes that are of the same order of magnitude. [Lenstra, 1987] In order to maintain the security of protocols based on the factorization complexity assumption, one also needs to avoid the situation where $E(\mathbb{F}_p)$ has a smooth order and one should want N to be a number with at least two large prime factors.

These are just a few examples of how researches tried decades to find a solution to these complexity problems and still the comprehensive solutions are not found.

Chapter 4

Basic implementations

We have seen some background information on cryptography and some of the concepts are about to be used. In this chapter the first implementations will be set out. No U-Prove will be involved yet; that will be the content of the next chapter. The protocols in this chapter are well known in the world of cryptography and are a good start to understand how protocols work. Three different types of cryptosystems will be discussed: *public-key*, *signature* and *proof of knowledge* cryptosystems. The protocols can be found in the chapter itself, though the implementations can be found at the back of the thesis as appendices.

4.1 Public-key cryptosystems

These systems make use of two different keys for encryption; a public key and a private key. Obviously, the public key is known by everyone and the private key is only known by Alice and Bob. Public key cryptosystem, where one needs a different key to encrypt than to decrypt, is an *asymmetric* cryptosystem. The message Bob wants Alice to know is called the *plaintext*. So pay attention: Bob is sending a message to Alice, not the other way around. The encrypted version of the plaintext that is actually sent, is called the *ciphertext*. The two protocols in this section are both encryption protocols, but also the decryption is part of it. Otherwise, it would of course not make sense to perform the protocol.

4.1.1 ElGamal encryption

The ElGamal encryption protocol [ElGamal, 1985] works in a multiplicative group G with generator g; g has prime order p. Alice chooses a private key x uniformly at random from $\mathbb{F}_p$. Choosing an element randomly will be denoted as $\in_{\mathcal{R}}$ in protocols.

Alice calculates her public key h with g and x, as can be seen in protocol 4.1, and sends it to Bob. He has a message $m \in G$ he wants to send to Alice. He chooses an element y uniformly at random from $\mathbb{F}_p$. The ciphertext consists of two parts, c_1 and c_2. Bob creates these two parts of the ciphertext and sends them both to Alice. She does not know y , but this is not a problem since she is the only one who knows x. The ciphertext can be decrypted by Alice to m with $c_2(c_1^x)^{-1}$, where $(c_1^x)^{-1}$ is the inverse of c_1^x.

Public information: multiplicative group G, generator g, $ord(g)$ is prime p.

Alice		**Bob**
$x \in_{\mathcal{R}} [0, ..., p-1]$		
$h := g^x$		
	$\xrightarrow{h}$	
		$m \in G$
		$y \in_{\mathcal{R}} [0, ..., p-1]$
		$c_1 := g^y$
		$c_2 := mh^y$
	$\xleftarrow{(c_1, c_2)}$	
$m = c_2(c_1^x)^{-1}$		

PROTOCOL 4.1: ElGamal encryption.

Eve is not able to find the plaintext m, though she can intercept the elements that are sent through the public channel. There we find a downfall of this encryption protocol. When Eve intercepts, for example, (c_1, c_2) and multiplies the second component by α then Alice will not decrypt m but αm. There is no way to avoid this issue with this protocol. The implementation of the protocol can be found in appendix A. There is a limitation to the implementation. When the multiplicative group is too big, i.e. p is chosen too large, the time limit of two minutes is exceeded. The prime p can be chosen between 0 and numbers up to around 2^{299}.

4.1.2 Paillier encryption

The protocol of the Paillier encryption [Paillier, 1999] is built around the group $(\mathbb{Z}/n^2\mathbb{Z})^*$, which will be denoted as G. Here n is the product of two large primes p_1 and p_2, chosen by Alice. Also an element g is chosen from G such that the order of g is divisible by n.

See protocol 4.2 for the protocol overview. First Alice picks these p_1 and p_2 and sends n through the public channel to Bob. In turn, Bob chooses a ρ uniformly at random from $(\mathbb{Z}/n\mathbb{Z})^*$ and encrypts the plaintext $m \in [0, n)$ he wants to send to Alice. In order to do this he first needs to find the smallest representative $\tilde{\rho}$ of ρ in G, otherwise the multiplication to obtain ciphertext c is not defined. Though when it comes to implementing this in Magma is easily done with the command `rhotilde := Zn2!rho`. This command states `rhotilde` is the element `rho` in G; see appendix B for the whole implementation. In [Weitenberg, 2012] this is explained more extensively. Then Bob send c to Alice and now she can decrypt to obtain the plaintext.

Private information of Alice: two large prime p_1 and p_2, she makes $n := p_1 p_2$ publicly known.
Public information: group $G = (\mathbb{Z}/n^2\mathbb{Z})^*$, generator g, $ord(g)$ must be divisible by n.

Alice		**Bob**
		$m \in [0, n)$
		$\rho \in_{\mathcal{R}} (\mathbb{Z}/n\mathbb{Z})^*$
		Find $\tilde{\rho}$ of ρ in G
		$c := g^m \tilde{\rho}^n \bmod n^2$
	$\xleftarrow{c}$	
$\lambda := lcm(p_1 - 1, p_2 - 1)$		
$m = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)}$		

Protocol 4.2: Paillier encryption.

Here L is defined as $L(x) := (x - 1)/n$. Like the ElGamal encryption protocol, this protocol has the same limitation: p_1 and p_2 must not be chosen too large, the time limit will be exceeded. The primes p_1 and p_2 can be chosen between 0 and numbers up to around 2^{109}.

The Paillier encryption protocol has two nifty properties which make this protocol useful for secure voting. Let c_i be the ciphertext of m_i (with $i \in \{1, 2\}$), than

- $c_1 c_2$ decrypts to $m_1 + m_2$;
- c_1^k decrypts to $k m_1$.

Consider a voting where one should vote for or against a certain statement. Every voter should be able to cast a vote anonymous, so the person who counts the votes may not know whether someone votes 1 ('for') or 0 ('against'). Each voter will encrypt the vote with the Paillier encryption protocol and passes it over to him. He will eventually have t votes. He will multiply the ciphertexts, $c_1 c_2 \dots c_t$, and decrypt it to $s := m_1 + m_2 + \dots + m_t$. Now he knows s people have voted 'for' and $t - s$ people have voted 'against', but he does not know who voted what.

4.2 Signature cryptosystem

The purpose of a signature cryptosystem is to prove the authenticity of the message that is sent. Alice sends Bob a message m and to prove she is the one sending it to him, she signs it with a signature. Bob checks if the signature is valid and accepts it or rejects it. On the other hand Alice can deny she sent this message but now Bob has the proof that she did, while her private key stays secret.

4.2.1 ElGamal signature scheme

Here is an example of a signature cryptosystem, namely the ElGamal signature scheme [ElGamal, 1985]. Here Alice signs her message m with a signature (s_1, s_2). The signature scheme only signs a message and verifies the signature. In order to encrypt and sign a message one should use the ElGamal encryption to encrypt and the signature scheme to sign it. Another important issue is the usage of a collision resistant hash function $\mathcal{H}$. The concept of a hash function is explained in the previous chapter. A hash function is collision resistant when it is hard to find messages m_1 and m_2 such that $\mathcal{H}(m_1) = \mathcal{H}(m_2)$.

See protocol 4.3 for the exact scheme. Again Alice picks a private key x and creates a public key $h := g^x$. Then, uniformly at random, she chooses y between 0 and $p-1$, such that the greatest common divisor of y and $p-1$ equals 1. The last two steps encompass the creation of the two parts of the signature (s_1, s_2). The second part must not be equal to 0, if so, one should start again at the step where y is chosen.

Public information: multiplicative group $G = \mathbb{F}_p^*$ with generator g.

Alice		**Bob**
$x \in_{\mathcal{R}} [0, ..., p-1]$		
$h := g^x \bmod p$		
$m \in G$		
$\star$		
$y \in_{\mathcal{R}} (0, p-1)$, with $gcd(y, p-1) = 1$		
$s_1 := g^y \bmod p$		
$s_2 := (\mathcal{H}(m) - xs_1)y^{-1} \bmod (p-1)$		
if $s_2 = 0$ then start again at $\star$		
	$\xrightarrow{(h, \mathcal{H}(m), s_1, s_2)}$	
		$0 < s_1 < p$
		$0 < s_2 < p-1$
		$g^{\mathcal{H}(m)} \stackrel{?}{\equiv} h^{s_1} s_1^{s_2} \bmod p$

Protocol 4.3: ElGamal signature scheme.

The security of the signature scheme depends on the security of the hash function. If the hash function is secure than so is the signature scheme. The implementation can be viewed in appendix C. The prime p can be chosen really large; between 0 and numbers up to around 2^{396}.

4.3 Proof of knowledge cryptosystems

The name of this kind of cryptosystem already reveales it; the goal is to prove one has knowledge of something. The answer to the question 'why would one want that?' gives the essence of these cryptosystems. One proves knowledge of something without revealing the content of the knowledge. Otherwise, it would be nonsense; Alice says: 'I know the key to break the system, because here it is!' and then everyone knows the key and the system will be useless. To make sure this is not the only way to convince someone you know a certain thing, *proof of knowledge* cryptosystems were invented.

In this section the two parties communicating are no longer Alice and Bob but names that reminds one better of there purpose, this differs per subsection. Proof of knowledge protocols consist of three parts: the commitment, the challenge and the response. Of course, the final step will always be the verification, but that is something only for the Verifier. The Verifier verifies there is proof of knowledge. The three parts are pieces of information that are generally sent through the public channel. Remark: from now on elliptic curves are involved!

4.3.1 Schnorr proof of knowledge

With the Schnorr proof of knowledge [Schnorr, 1989] the Prover tries to prove knowledge over the private key x. Saying 'the Prover proves knowledge over x' means the Prover proves he knows x. The scheme can be viewed in protocol 4.4. Remember the complexity assumption ECDLP, so it is indeed hard for the Verifier to find x.

The prime q is the order of the cyclic group $\{P, [2]P, [3]P, ...\} \subseteq \mathbb{E}(\mathbb{F}_p)$. First ω is chosen uniformly at random from $[0, ..., q-1]$ by the Prover and with that the commitment W is calculated and sent to the Verifier. The Verifier chooses a challenge γ also from $[0, ..., q-1]$ uniformly at random and sends it to the Prover. In turn, the Prover calculates the response $r := \gamma x + \omega \bmod q$ and sends this to the Verifier. The Verifier can now verify if the Prover has knowledge of x by checking whether W equals $rP - \gamma X$.

The implementation of this protocol can be found in appendix D. There must be made some remarks to this, because three parts might not be that obvious. First, one must be

Public information: Elliptic curve group $E(\mathbb{F}_p)$ of prime order q, generated by point P, prime p and point $X \in E(\mathbb{F}_p)$, a multiple of P.
Private information of the Prover: $x \in [0, ..., q-1]$, such that $X = xP$.

Prover		**Verifier**
$\omega \in_{\mathcal{R}} [0, ..., q-1]$		
$W := \omega P$		
	$\xrightarrow{W}$	
		$\gamma \in_{\mathcal{R}} [0, ..., q-1]$
	$\xleftarrow{\gamma}$	
$r := (\gamma x + \omega) \bmod q$		
	$\xrightarrow{r}$	
		$W \stackrel{?}{=} rP - \gamma X$

PROTOCOL 4.4: Schnorr proof of knowledge.

certain to work with an elliptic curve that is not singular, so $-16(4a^3 + 27b^2) \neq 0$ must hold. The second part is the while-loop when `IsPrime(#E) ne false`. This makes sure one gets an elliptic curve where the order of E is prime; with the result that $E(\mathbb{F}_p)$ is cyclic. Another thing to point out is the while-loop when `Order(P) ne q`. As a result one can be sure P is indeed a generator of the group $E(\mathbb{F}_p)$. The prime p may not be too large. With high probability the program still runs adequately when one takes a prime between 0 and 2^{191}.

4.3.2 Schnorr signature scheme

The scheme of this protocol [Schnorr, 1990] looks a lot like the scheme of the Schnorr proof of knowledge, but is has definitely an other purpose. Here the Signer, the one creating the signature, wants to prove that if a person knows x, this person will accepts the message m when the signature is valid. In other words, when a person knows x and the signature is valid then this person will assume the message m is sent by whom the sender claims to be. The scheme of this protocol is protocol 4.5. In this scheme the Verifier is the person who verifies the claim of the Signer. The message m is thus known to everyone, otherwise the claim cannot be verified. Here m is a string of zeros and ones of length δ.

The difference between the Schnorr proof of knowledge and this protocol is that the challenge γ is now hashed by the person who generates the commitment, the Signer, and not by someone else. After the signature (γ, r) is calculated it is sent for verification. This verification also includes the hash function; this is the consequence of hashing the challenge.

Public information: Elliptic curve group $E(\mathbb{F}_p)$ of prime order q, generated by point P, prime p and point $X \in E(\mathbb{F}_p)$, a multiple of P and the message $m \in \{0,1\}^\delta$.
Private information of the Signer: $x \in [0, ..., q-1]$, such that $X = xP$.

Signer		**Verifier**
$\omega \in_{\mathcal{R}} [0, ..., q-1]$		
$W := \omega P$		
$\gamma := \mathcal{H}(m\|\|W)$		
$r := (\gamma x + \omega) \bmod q$		
	$\xrightarrow{(\gamma, r)}$	
		$\gamma \stackrel{?}{=} \mathcal{H}(m\|\|rP - \gamma X)$

PROTOCOL 4.5: Schnorr signature scheme.

Implementing this protocol was not as easy as the previous ones. The reason for this is the symbol in the hash function, together with the fact that elliptic curves are used. The || stands for the concatenation of the element on the left and the element on the right side of this symbol. When one concatenates two elements one simply 'pastes' them together in the order they are noted. (Example: *crypto||graphy* equals *cryptography*.) As one can see m is twice concatenated with a point in $E(\mathbb{F}_p)$. One cannot 'past' them together, because m is a string of zeros and ones and a point on the chosen elliptic curve is something with two coordinates. Now look at the case of $\mathcal{H}(m||W)$, the second case is hashed in a similar way. The whole implementation can be found as appendix E.

To concatenate m and W and hash the contatenation, the following solution was found. Below the commands for this solution are displayed.

```
m := IntegerToString(m);
HashW:= Hash(W);
hashWstring := IntegerToString(HashW);
mcathashW := m cat hashWstring;

gamma := Hash(mcathashW);
```

First one must know, a point on an elliptic curve can be hashed to a single integer in Magma. This is how W was transformed into an integer. The command in Magma to concatenate two elements is `cat`. This can only be applied to two strings. After transforming `m` and `Hash(W)` into strings one can concatenate them. Now γ can be calculated as `Hash(mcathashW)`. Eventually the extra operation to hash $m||W$ is the hashing of W.

The prime p can, again, not be too large. With a prime between 0 and approximately 2^{182} the program can still run all the commands.

4.3.3 Schnorr blind signature scheme

In the previous subsection the signature scheme that is discussed allowed everyone to know message m and the signature (γ, r). This is not what one always wants. It is much more convenient when the Signer, for example some authority, never sees the message that is signed. Or even better, also never sees the signature itself. To go back to the voting example; it is not appropriate when the government sees what one is voting in order to confirm its legitimacy. Therefore, the *blind* signature schemes were introduced. The blindness of the scheme refers to the fact that the Signer is signing a message he/she does not know with a signature he/she never sees. In this subsection the extension of the Schnorr signature scheme, where it becomes a blind signature scheme, is discussed [Pointcheval & Stern, 1996]. Below, in protocol 4.6, the scheme is displayed.

The protocol is similar to the two previous ones. However two parameters, α and β, are added to blind the commitment, challenge and the response. A parameter with a $\sim$ is the blinded version of the parameter without it. The Receiver blinds the commitment received from the Signer by adding $\alpha X + \beta P$ to it. het also blinds γ, obtained from the blinded commitment and message m, by adding α before sending it to the Signer. The Signer calculates the response r and sends it to the Receiver. The Receiver checks if the response is valid and blinds the response by adding β to it. The Signer knows the blinded version of the challenge and the response, but does not know the challenge and the blinded version of the response. Therefore, the signature will be $(\gamma, \tilde{r})$.

Public information: Elliptic curve group $E(\mathbb{F}_p)$ of prime order q, generated by point P, prime p and point $X \in E(\mathbb{F}_p)$, a multiple of P.
Private information of the Signer: $x \in [0, ..., q-1]$, such that $X = xP$.
Private information of the Receiver: message $m \in \{0,1\}^\delta$.

Signer		**Receiver**
$\omega \in_{\mathcal{R}} [0, ..., q-1]$		
$W := \omega P$		
	$\xrightarrow{W}$	
		$\alpha, \beta \in_{\mathcal{R}} [0, ..., q-1]$
		$\widetilde{W} := W + \alpha X + \beta P$
		$\gamma := \mathcal{H}(m \| \widetilde{W})$
		$\tilde{\gamma} := (\gamma + \alpha) \bmod q$
	$\xleftarrow{\tilde{\gamma}}$	
$r := (\tilde{\gamma} x + \omega) \bmod q$		
	$\xrightarrow{r}$	
		$W \stackrel{?}{=} rP - \tilde{\gamma} X$
		$\tilde{r} := (r + \beta) \bmod q$

PROTOCOL 4.6: Schnorr blind signature scheme.

Implementing this protocol goes in a similar way as the Schnorr signature scheme. To hash a point on an elliptic curve one need to hash twice as explained in the previous subsection. The Magma-code can be viewed in appendix F. One cannot choose p too large; primes between 0 and 2^{172} will probably result in a complete run of the program.

Chapter 5

U-prove protocols

Hopefully the reader now has an idea about how protocols work and understands it is indeed possible to prove 'something' without revealing that 'something'. We have come to the point where we have all the instruments we need to set out the explicit U-prove protocols. To achieve that chapter 3 and 4 will be combined. In chapter 3 the reader got acquainted with credentials and already some insight of U-prove itself. From chapter 4 the reader should not forget the idea of the Schnorr blind signature scheme.

In chapter 3 a way to create the attribute commitment was demonstrated: $C = k_1X_1 + k_2X_2$, where X_1 and X_2 are two points on an elliptic curve and k_1 and k_2 are the values belonging to two attributes. One who shows the attribute commitment to someone, needs to prove he/she also knows the attribute values, but definitely not reveal them! U-prove combines this with the Schnorr blind signature scheme. This way the Issuer will never see the Receiver's credential and thus never know to whom the Receiver shows it to. [Weitenberg, 2012]

As you may have noticed the title of this chapter is called U-prove protocol*s*. The U-prove system consists of two protocols [Brands, 2000]. The *Issuing protocol* and the *Showing protocol*; each protocol has its own section in this chapter. The Issuing protocol deals with issuing a blind signature to the Receiver from an authority. The Showing protocol encompasses showing a part of the Receiver's credential to someone. U-prove in its entirety is receiving a blind signature from an authority to create a credential and proving one has a credential without presenting the entire credential itself.

5.1 Issuing protocol

To issue the blind signature one needs two parties: an authority to issue the signature, the *Identity Provider*, and the person who afterwards owns the signature and can use it to make a credential with it, the *User*. Before one can start the protocol the Identity Provider needs to choose some elements and agree on some with the User. First the elliptic curve group $E(\mathbb{F}_p)$ needs to be chosen together with a generator P. The order of P is denoted as q. The next step is deciding how many attributes a credential should contain, the number of attributes per credential is m. The above-described elements are broadcasted. After that the Identity Provider picks uniformly at random $y, x_i \in (\mathbb{Z}/q\mathbb{Z})^*$, $i \in \{1, 2, \ldots, m\}$; these are not broadcasted! With these values $m+1$ points in the elliptic curve group are found: $X_i = x_i P$ for all i and $Y = yP$. Next all the X_i's and Y are made public. The Identity Provider agrees with the User on the attribute values k_i.

Remember a credential is a signature from the Identity Provider together with the attribute commitment C. The User does not want the Identity Provider to know C as discussed above. The attribute commitment is something the User can generate on its own, the missing piece of the credential is the signature. Also the signature must be a secret to the Identity Provider. So when one follows the scheme of this protocol (see protocol 5.1) the result will be a blind signature for C, similar to the Schnorr blind signature scheme.

As in the previous chapter the commitment W is generated. The Identity Provider chooses ω uniformly at random from $\mathbb{Z}/q\mathbb{Z}$, followed by sending the commitment $W := \omega P$ to the User. The User chooses $k_0, \alpha, \beta \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$, where k_0 is the private key of the User and α and β are the parameters to blind the two elements of the signature issued by the Identity Provider. Then the User calculates his/her attribute commitment $C := k_0 P + C_U$ and blinds the commitment W. Now the challenge is defined as $\gamma := \mathcal{H}(C||\widetilde{W})$ and after blinding it with α, $\tilde{\gamma}$ is sent to the Identity Provider. In turn, the Identity Provider calculates the response r and sends it to the User. The User verifies whether the response is valid and blinds this response to obtain the final signature $(\gamma, \tilde{r})$ belonging to the attribute commitment C.

In this protocol one does not hash the concatenation of an integer and an element of the elliptic curve group. The concatenation of two elements from the elliptic curve group is hashed. In order to hash $C||\widetilde{W}$, first C and $\widetilde{W}$ need to be hashed separately. This does not result into many problems. Furthermore, one must hold on to $\tilde{r}$ without the mod q. Still, in the second part of the signature the mod q is included as the protocol shows. The $\tilde{r}$ without the mod$\,q$ is needed for the Showing protocol. Otherwise the verification in the beginning will be stated invalid. The implementation can be found in appendix

Public information: Elliptic curve group $E(\mathbb{F}_p)$, generated by point P, $q = ord(P)$, prime p, points Y, $\{X_i\}_{i=1}^m \in E(\mathbb{F}_p)$ (all multiples of P), the attribute values $\{k_i\}_{i=1}^m$ and $C_U = \sum_{i=1}^m k_i X_i$.
Private information of the Identity Provider: $y, x_i \in (\mathbb{Z}/q\mathbb{Z})^*$, such that $X_i = x_i P \ \forall i$ and $Y = yP$.

Identity Provider		**User**
$\omega \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$		
$W := \omega P$		
	$\xrightarrow{W}$	
		$k_0 \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$
		$\alpha, \beta \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$
		$C := k_0 P + C_U$
		$\widetilde{W} := \alpha(Y + C_U) + \beta P + W$
		$\gamma := \mathcal{H}(C \| \widetilde{W})$
		$\tilde{\gamma} := \gamma + \alpha \bmod q$
	$\xleftarrow{\tilde{\gamma}}$	
$r := \omega + \tilde{\gamma}(y + \sum_{i=1}^m k_i x_i)$		
	$\xrightarrow{r}$	
		$W \stackrel{?}{=} rP - \tilde{\gamma}(Y + C_U)$
		$\tilde{r} := r + \beta + \gamma k_0 \bmod q$

PROTOCOL 5.1: U-prove: Issuing protocol.

G. The attribute values k_i are placed in a vector of length m. The same is done for the x_i's. As an example $\{k_1, k_2, \ldots, k_m\}$ is chosen to be $\{1, 2, \ldots, m\}$ and m equals 10. As one can see in the appendix the prime p can be chosen really large; between 0 and 2^{339}.

Now the User has a valid signature $(\gamma, \tilde{r})$ for attribute commitment C. Both the elements of the signature are unknown to the Identity Provider, together with C they form the credential.

5.2 Showing protocol

The Showing protocol can only be executed when the Issuing protocol is already performed, because the signature obtained from the Issuing protocol is needed. The User has finished the communication with the Identity Provider. Now the User is going to prove to a Verifier that a part of the attribute values belong the attribute commitment signed by the Identity Provider and is the one whom the Identity Provider assigned it to. To prove the first statement the User only needs to show a few attribute values and not all of them. To prove the signature is assigned to the User, the User sends the signature to the Verifier for verification. The hidden attribute values (only known

to the User) are indexed by $\mathcal{C}$. As a result the public attribute values are indexed by $\mathcal{D} = \{1, 2, \ldots, m\} \setminus \mathcal{C}$. The scheme of this protocol can be found below in protocol 5.2.

The verification of the signature $(\gamma, \tilde{r})$ is the first step. So the User sends the signature to the Verifier who verifies it. (This is the verification mentioned in the previous section where $r+\beta+\gamma k_0$ is needed instead of $r+\beta+\gamma k_0 \bmod q$, otherwise the Verifier will reject the signature.) The User calculates the commitment $\widehat{W}$ with the $\#\mathcal{C}+1$ omega's that are chosen uniformly at random from $\mathbb{Z}/q\mathbb{Z}$ and sends it to the Verifier. The challenge $\hat{\gamma}$ is chosen uniformly at random from $(\mathbb{Z}/q\mathbb{Z})^*$ and sent to the User. The response $r_0 \cup \{r_i\}_{i\in\mathcal{C}}$ is then calculated by the User and completely sent to the Verifier. The Verifier defines $C_{\mathcal{C}} := C - C_{\mathcal{D}}$ and checks whether this $C_{\mathcal{C}}$ indeed provides a proof of knowledge.

Public information: Elliptic curve group $E(\mathbb{F}_p)$, generated by point P, $q = ord(P)$, prime p, points Y, $\{X_i\}_{i=1}^m \in E(\mathbb{F}_p)$ (all multiples of P), a part of the attribute values $\{k_i\}_{i\in\mathcal{D}}$, $C_{\mathcal{D}} = \sum_{i\in\mathcal{D}} k_i X_i$ and the index sets $\mathcal{C}$ and $\mathcal{D}$.
Private information of the User: Hidden attribute values $\{k_i\}_{i\in\mathcal{C}}$, the attribute commitment C and the signature $(\gamma, \tilde{r})$.

User		**Verifier**
	$\xrightarrow{C,(\gamma,\tilde{r})}$	
		$\gamma \stackrel{?}{=} \mathcal{H}(C\|\|\tilde{r}P - \gamma(C+Y))$
$\omega_i \in_{\mathcal{R}} \mathbb{Z}/q\mathbb{Z}$, $\forall i \in \{0\} \cup \mathcal{C}$		
$\widehat{W} := \omega_0 P + \sum_{i\in\mathcal{C}} \omega_i X_i$		
	$\xrightarrow{\widehat{W}}$	
		$\hat{\gamma} \in_{\mathcal{R}} (\mathbb{Z}/q\mathbb{Z})^*$
	$\xleftarrow{\hat{\gamma}}$	
$r_0 := \hat{\gamma} k_0 + \omega_0$		
$r_i := \hat{\gamma} k_i + \omega_i, \quad \forall i \in \mathcal{C}$		
	$\xrightarrow{r_0,\{r_i\}_{i\in\mathcal{C}}}$	
		$C_{\mathcal{C}} := C - C_{\mathcal{D}}$
		$\widehat{W} + \hat{\gamma} C_{\mathcal{C}} \stackrel{?}{=} r_0 P + \sum_{i\in\mathcal{C}} r_i X_i$

Protocol 5.2: U-prove: Showing protocol.

In appendix H the implementation can be found. The index set $\mathcal{D}$ is chosen to be $\{1, 3, 5, 7, 9\}$. As a result the index set $\mathcal{C}$ equals $\{2, 4, 6, 8, 10\}$. So five attribute values, k_1, k_3, k_5, k_7 and k_9, are public and the other five are hidden. Prime p can be chosen between 0 and 2^{339}.

It is important to mention: the Identity Provider and the Verifier do not know they have communicated with the same User, because the Identity Provider never sees the signature.

Chapter 6

Extension of U-prove

We are getting to the core of Weitenberg's [2012] research; the extension of the U-prove protocols. One should make sure the concept of U-prove is completely clear before reading this chapter, because it will become more specific and detailed. Why should U-prove be extended? There is a disadvantage to the protocols from the previous chapter. This has everything to do with unlinkability. When two protocols are unlinkable one cannot determine whether the protocols are generated by the same User. To state it more formally:

Definition (Unlinkability). Two protocols are *unlinkable* when they cannot be linked to each other, i.e. there does not exist an adversary who can determine, with higher probability than $\frac{1}{2}$, whether the protocols are a communication with the same User.

An adversary is a common term for the attacker. The negation of unlinkable is linkable. Weitenberg rightly concludes the Issuing protocol and Showing protocol are unlinkable when all the attributes are hidden. He states it rather obvious: 'If you disclose information that might identify you, you shouldn't be surprised when you are indeed identified'. The problem that arises and the reason of Weitenberg's research, is the linkability of the Showing protocol to itself. The User cannot blind the attribute commitment and the signature, because the signature then becomes invalid. When a User does not want people to know two or more credentials are both from him, he needs to destroy every credential after using it. Each time the User uses a credential, he needs a new credential. This is not ideal, because the Issuing protocol must be executed each time a User needs a new credential. Furthermore, smart cards have storage limitations, so refreshing your credential each time will eventually cause problems. [Weitenberg, 2012]

Weitenberg therefore designed an extension to the U-prove protocols such that the Showing protocol becomes unlinkable to itself. This means a User should be capable of blinding the credentials himself. This way, a User can use a credential multiple times. In this case *blinding* means adding a uniformly at random chosen parameter (the blind) to the element one wants to blind. The result appears uniformly at random as well to anyone who does not know the blind. Also the signature is still valid for the attribute commitment. Blinding is not the only way to achieve unlinkability, but this simply is what Weitenberg did. Above all the advantage of U-prove, the ignorance of the Identity Provider and the Verifier whether they communicated with the same User, still holds.

In this chapter we will not discuss the security of the extension protocols; more can be read about this in Weitenberg's thesis, this is not our focus. Our focus lies on implementing the protocols in Magma. As mentioned before pairings are used as a black box in Weitenberg's thesis. In order to implement theory on pairings and an explicit pairing are needed. In the first section we will give an outline of the situation where the extension protocols are described. We will conclude the missing piece of knowledge is the pairing. Therefore, in the second section pairings will be explained and the most famous pairing will be set out, the Weil pairing. A solution to finding the right pairing and an example of implementing it will be presented in the last section. Thereafter in two subsequent subsections, two other attempts of implementing the alternative pairing are described. Unfortunately, I did not succeed, though it is valuable to examine what happens. In the final subsection the extention protocols are implemented with the example for which I was able to implement the alternative pairing.

6.1 Outline of the situation

The goal is to construct a signature that remains valid for C after blinding, but an invalid one after other transformations like changing some attribute values. This has led Weitenberg to use a signature scheme from Boneh and Boyen [2008], which includes a pairing. Pairings will be discussed in the next section, but Weitenberg gives a short but essential reasoning why he chose the signature (S, r). (For now, assume a pairing is a transformation that maps two points from elliptic curve groups to an integer.) The reasoning goes as follows.

The first part of the signature is computed as:

$$S = \frac{1}{y + c + rz} \cdot Q,$$

because this is easy to verify with a pairing. Here S is the first part of the constructed signature, y, c and z are discrete logarithms such that $Y = yP$, $C = cP$ and $Z = zP$ (defined as in the previous chapter), Q is the generator of another elliptic curve group than the one generated by P and r is chosen uniformly at random from $\mathbb{Z}/q\mathbb{Z}$, where q is the order of P. The signature can be verified by

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

where e denotes the pairing. The signature can be blinded safely by multiplying a random scalar from $(\mathbb{Z}/q\mathbb{Z})^*$ and therefore obtain unlinkability. To blind the commitment of the showing part, the User chooses a ν uniformly at random from $(\mathbb{Z}/q\mathbb{Z})^*$ and computes $\tilde{C} = C + \nu Z$ and $\tilde{r} = r - \nu$. Since

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

$(S, \tilde{r})$ is a valid signature for attribute commitment $\tilde{C}$. The User is still recognizable by the value $\tilde{C} + \tilde{r}Z$ and S itself, so the two parts of the signature $(S, \tilde{r})$ have to be blinded separately. This last step is done by multiplying with a blind. [Weitenberg, 2012]

The above is a rough sketch of why these modifications are chosen and how the extension protocols work. The extension protocols are, as one may expect, more complicated than the original protocols. Blinding takes some more steps and the verifications in between do too.

6.1.1 Extension Issuing protocol

The difference between the original Issuing protocol and this extended version is mainly the setup. The new Issuing protocol is created such that it matches the new Showing protocol perfectly. The general outline is the same as the original issuing protocol; a blind signature is created and verified.

This setup involves two elliptic curve groups. The restrictions of these groups are presented in the next section, because it has everything to do with pairings. The scheme of this protocol is displayed in protocol 6.1 and as one can see the second elliptic curve group E_2 is not defined over a set. The reason for this are these restrictions. Also the notation of the attribute commitment $C = k_0X_0 + \sum_{i=1}^{m} k_iX_i$ might appear strange, we do this because k_0 is the User's private key ans the k_i's are attribute values. Another thing to mention are the private keys of the Identity Provider. In Weitenberg's thesis these private keys x_i, y and z are said to be chosen from $\mathbb{F}_q$, but I was not sure about

this. If one of these values is zero, multiplication with P would be the element at infinity. The multiple of P is public information and in this case q, the order of P, must be prime. So one can easily conclude the discrete logarithm is 0 mod q. This is why I choose them from $\mathbb{F}_q^*$ to exclude the zero.[1] One last remark must be made about this protocol. The *Enc* and *Dec* stand, respectively, for encryption and decryption using the Paillier encryption.

Now we have come to the protocol itself. The User and Identity Provider both blind the items they send in consideration of their private keys. First the User creates a Paillier setup as shown in chapter 4 with one of the prime factors of n equal to q [Weitenberg, 2012]. The User keeps λ private for the decryption at the end of the protocol. After the Identity Provider receives $\bar{b}$, $\bar{r}$ and $\bar{k_0}$, the signature is computed within the Paillier encryption and is sent to the User in the form of d [Hoepman & Lueks, 2012]. To check whether the communication has gone as agreed (as described in the protocol) the User wants to check d. This check is done by a proof of knowledge so this is a bit of a sidestep.

When the User verifies d is indeed created correctly, he decrypts d and sends the (still blinded) decryption $\bar{s}$ to the Identity Provider. The Identity Provider removes the blind by transforming $\bar{s}$ into s, computes $\tilde{S}$ and sends it to the User. The User unblinds $\tilde{S}$ by multiplying it with his private key β. Now both parts of the blind signature are obtained by the User, who checks whether (S, r) is indeed a valid signature for attribute commitment C.

Again, the safety of this protocol lies not the scope of this thesis. For the more advanced reader, Weitenberg proved this protocol is witness-indistinguishable with respect to the Users private key k_0.

6.1.2 Extension Showing protocol

The setup of the extended Showing protocol is similar to that of the original Showing protocol. Again, the User shows some of the attributes of his credential and proves there legitimacy by using the signature obtained from communication with the Identity Provider, the Issuing protocol. Though this extended version blinds certain elements in order to randomise the attribute commitment and the signature. Quite logical, because to obtain unlinkability the User must never tell the Verifier the attribute commitment and the signature itself.

As told before, points in elliptic curve groups can be blinded by multiplying them with an element from $\mathbb{F}_q^*$. This can be done to blind the first part S, but not to blind r,

[1] I contacted Erik Weitenberg. He and his friend Wouter Lueks confirmed this is a safer choice.

Public information: Elliptic curve group $E_1(\mathbb{F}_p)$, generated by point P, prime p, order of P is prime q, elliptic curve group E_2, generated by point $Q \notin E_1(\mathbb{F}_p)$, points Y, $Z \in E_1(\mathbb{F}_p)$ (multiples of P) and the attribute values $\{k_i\}_{i=1}^m$ all in $\mathbb{F}_q$.
Private information of the User: Private key $k_0 \in \mathbb{F}_q$, attribute commitment $C = k_0X_0 + \sum_{i=1}^m k_iX_i$.
Private information of the Identity Provider: $x_i, y, z \in \mathbb{F}_q^*$, such that $X_i = x_iP \ \forall i$, $Y = yP$ and $Z = zP$.

Identity Provider		**User**
		Create Paillier setup (n, λ, g)
	$\xleftarrow{n,g}$	
		$\beta, r \in_{\mathcal{R}} [0, n)$ $\bar{b} := Enc(\beta)$ $\bar{r} := Enc(\beta r)$ $\bar{k_0} := Enc(\beta k_0)$
	$\xleftarrow{\bar{b},\bar{r},\bar{k_0}}$	
$\gamma \in_{\mathcal{R}} \mathbb{F}_q^*$ $k := y + \sum_{i=1}^m k_i x_i$ $d := \bar{r}^z \bar{k_0}^{x_0} \bar{b}^k Enc(\gamma) \bmod n^2$		
	$\xrightarrow{d}$	
		Check with interactive proof of knowledge if d was constructed as shown above. $\bar{s} := Dec(d) \bmod n$
	$\xleftarrow{\bar{s}}$	
$s := \bar{s} - (\gamma \bmod q) \bmod q$ $\tilde{S} := (s^{-1} \bmod q)Q$		
	$\xrightarrow{\tilde{S}}$	
		$S := \beta\tilde{S}$ $e(Y + C + rZ, S) \stackrel{?}{=} e(P, Q)$

Protocol 6.1: Extension U-prove: Issuing protocol.

therefore Weitenberg changed the verification a little bit. The new verification will be:

$$e(Y + \tilde{C}, \tilde{S})e(Z, \tilde{R}) \stackrel{?}{=} e(P, \tilde{Q}), \qquad \text{with } \tilde{R} := \tilde{r}\tilde{S}.$$

Like the discrete logarithms in the extension of the Issuing protocol, I changed the set where γ is chosen from; again excluding zero compared to Weitenberg's description. The reason, I presume, is quite obvious when one looks to the scheme of this protocol in protocol 6.2. (The element γ is known to everyone; when γ equals zero the users private keys are revealed.)

As the first step of the protocol the User blinds the attribute commitment with ν, sends $\tilde{C}$ to the Verifier and discloses the attribute values with index set $\mathcal{D}$. Thereafter the

User generates $\sigma \in_{\mathcal{R}} \mathbb{F}_q^*$ and blinds the signature and sends it to the Verifier, who in his turn verifies this blinded signature. With a sidestep, Schnorr proof of knowledge, the Verifier checks whether the User indeed knows σ and $\tilde{r}$ such that $\tilde{Q} = \sigma Q$ and $\tilde{R} = \tilde{r}\tilde{S}$. After this proof of knowledge the User proofs knowledge over all the attribute values. A commitment N, a challenge γ and a response $r_\nu, r_{k_0}, \{r_i\}_{i \in \mathcal{C}}$ are created and blinded (except the challenge) and sent back and forth. The final step is the verification executed by the Verifier to check whether there is a proof of knowledge over all the attribute values. The definition of N and the last verification differ from what is written in Weitenberg's thesis, this is due to a typo; here P is replaced by Z.

We completed the entire U-prove extension protocols. Each time the User wants to use his credential, he blinds his attribute commitment C with a uniformly at random chosen ν and therefore randomizes C. Thus the system has now become unlinkable. One can assume blinding the attribute commitment and the signature is secure, but this is not proven by Weitenberg. So the security of the system remains as a question. For the enthusiasts: in Brands [2000] it was stated that the last proof of knowledge, starting at the line where k_0' is chosen, is witness-indistinguishable.

6.2 Pairings

It is not proven that the extension protocols are secure, i.e. there is no complexity assumption where these protocols can be linked to. Though Weitenberg does comment that obvious possibilities are assumptions about pairings, 'since the additive blinds ν occur in both elliptic curve groups'.

To go back to our main topic, implementing the procotols, we assume we can implement most of the two extension protocols like we did with the original U-prove. Likewise, we can pick random elements, add, multiply, etcetera. However there is a big difference: these protocols include a pairing and this is the obstacle that needs to be overcome. In this section we will discover what pairings are and the properties researchers find so interesting. Finally, the Weil pairing is set out and explained, because this pairing is playing a big part in the solution of implementing the extension protocols.

6.2.1 Theory and properties

Consider three groups, G_1, G_2 and G, all of prime order q. A *pairing* e is defined as

$$e : G_1 \times G_2 \longrightarrow G.$$

Public information: Elliptic curve group $E_1(\mathbb{F}_p)$, generated by point P, prime p, order of P is prime q, elliptic curve group E_2, generated by point $Q \notin E_1(\mathbb{F}_p)$, points $\{X_i\}_{i=1}^m$ Y, $Z \in E_1(\mathbb{F}_p)$ (multiples of P) and index sets $\mathcal{D}$ and $\mathcal{C}$.
Private information of the User: The attribute values $\{k_i\}_{i=1}^m$ all in $\mathbb{F}_q$, the attribute commitment C and the signature (S, r).

User		**Verifier**
$\nu \in_{\mathcal{R}} \mathbb{F}_q^*$		
$\tilde{C} := C + \nu Z$		
	$\xrightarrow{\tilde{C},\{k_i : i \in \mathcal{D}\}}$	
		$C_{\mathcal{D}} := \sum_{i \in \mathcal{D}} k_i X_i$
$\sigma \in_{\mathcal{R}} \mathbb{F}_q^*$		
$\tilde{S} := \sigma S$		
$\tilde{r} := r - \nu$		
$R := \tilde{r} S$		
$\tilde{R} := \tilde{r} \tilde{S}$		
$\tilde{Q} := \sigma Q$		
	$\xrightarrow{\tilde{S},\tilde{R},\tilde{Q}}$	
		$e(Y + \tilde{C}, \tilde{S}) e(Z, \tilde{R}) \stackrel{?}{=} e(P, \tilde{Q})$ $\wedge (\tilde{Q} \neq \mathcal{O})$ Interactive Schnorr proof of knowledge that User knows σ and $\tilde{r}$ such that $\tilde{Q} = \sigma Q$ and $\tilde{R} = \tilde{r} \tilde{S}$.
$k_0', \nu' \in_{\mathcal{R}} \mathbb{F}_q$		
$k_i' \in_{\mathcal{R}} \mathbb{F}_q \; \forall \; i \in \mathcal{C}$		
$N := \nu' Z + k_0' X_0$ $+ \sum_{i \in \mathcal{C}} k_i' X_i$		
	$\xrightarrow{N}$	
		$\gamma \in_{\mathcal{R}} \mathbb{F}_q^*$
	$\xleftarrow{\gamma}$	
$r_\nu := \gamma \nu + \nu'$		
$r_{k_0} := \gamma k_0 + k_0'$		
$r_i := \gamma k_i + k_i' \; \forall \; i \in \mathcal{C}$		
	$\xrightarrow{r_\nu, r_{k_0}, \{r_i\}_{i \in \mathcal{C}}}$	
		$N \stackrel{?}{=} r_\nu Z + r_{k_0} X_0 + \sum_{i \in \mathcal{C}} (r_i X_i)$ $-\gamma(\tilde{C} - C_{\mathcal{D}})$

PROTOCOL 6.2: Extension U-prove: Showing protocol.

For a fixed $g_1 \in G_1$ the map $g_2 \mapsto e(g_1, g_2)$ has to be a homomorphism, with $g_2 \in G_2$, and for a fixed g_2 the map $g_1 \mapsto e(g_1, g_2)$ has to be a homomorphism too.

It is possible that G_2 equals G_1. In the extension protocols the groups G_1 and G_2 stand, respectively, for the elliptic curve groups $E_1(\mathbb{F}_p)$ and E_2. As a reminder, this is indeed what happens in those protocols; the first entry is an element of $E_1(\mathbb{F}_p)$ and the second entry is an element of E_2 and e maps these two elements to an element of some group G.

From now on we will only consider the case where a pairing maps two points of elliptic curve groups to an integer, to stay close to our problem.

There are two properties that must hold in order to make a pairing suitable for applications in cryptography.

- *Bilinear.* $e(P_1 + P_2, Q) = e(P_1, Q)e(P_2, Q)$ and $e(P, Q_1 + Q_2) = e(P, Q_1)e(P, Q_2)$.
- *Non-degenerate.* If $e(P, Q) = 1 \ \forall \ P \in G_1$, then $Q = \mathcal{O}$.

Reseachers who are using pairings as a black box are using these properties in their protocols without using an explicit pairing itself.

Furthermore, a pairing can be symmetric or anti-symmetric. Let $\Pi : G_2 \to G_1$ and $\Sigma : G_1 \to G_2$ be two isomorphisms. A pairing is said to be symmetric when $e(\Pi(Q), \Sigma(P)) = e(P, Q)$ for all $(P, Q) \in G_1 \times G_2$ and anti-symmetric when $e(\Pi(Q), \Sigma(P)) = e(P, Q)^{-1}$ for all $(P, Q) \in G_1 \times G_2$.

Pairings are categorized in three different types. The security of a protocol also depends on the security of the pairing, so one must choose the type that most likely maintains this security.

Type I. $G_1 = G_2$.

Type II. $G_1 \neq G_2$ and there exists an efficiently computable homomorphism $\phi : G_1 \to G_2$.

Type III. $G_1 \neq G_2$ and there does not exist an efficiently computable homomorphism between G_1 and G_2.

According to Weitenberg the type needed for the extension protocols is type III; there must not exist an efficiently computable homomorphism between $E_1(\mathbb{F}_p)$ and E_2. He gives no reason for this choice, but I guess he thought it would be the most secure choice; taken in consideration that he did not explain the concept of a pairing.

His choice of a type III pairing can be justified. The security of a type I pairing is much more costly than the security of a type III pairing. And some claim a type II pairing is merely an inefficient implementation of a type III pairing. The only thing one has to sacrifice, when choosing a type III pairing, is the homomorphism from G_1 to G_2. [Costello, 2012]

6.2.2 Example: the Weil pairing

Over the years different pairings were described. All these pairings are based on one single pairing: the *Weil pairing*. The Weil pairing is also a pairing defined on elliptic curve groups and is bilinear, anti-symmetric and non-degenerate. The pairing that will be used to implement the extension protocols is also a modification of the Weil pairing. Enough reasons to discuss the Weil pairing.

Consider an elliptic curve group $E(K)$, with K a field and choose a prime ℓ different from the characteristic of K.

Definition (ℓ-torsion). The *ℓ-torsion* of E is defined as

$$E[\ell] = \{P \in E | \ell P = \mathcal{O}\}.$$

So the ℓ-torsion are all the points in $E(K)$ with an order that divides ℓ. With ℓ is prime, $E[\ell]$ consists of all points of order 1 and ℓ.

Now look at the splitting field L of $X^\ell - 1$ over K, i.e. the field extension L of K such that factors $X^\ell - 1$ into linear factors $\prod_{i=1}^{\ell}(X - \zeta_i)$ and $L = K(\zeta_1, ..., \zeta_\ell)$.

Definition (ℓ^{th} roots of unity). The *ℓ^{th} roots of unity* are defined as

$$\mu_\ell = \{\zeta_i | i = 1, \ldots, \ell\} \subset L^*.$$

Here it becomes clear why one needs to choose ℓ different from the characteristic of K, because than $\mu_\ell \cong \mathbb{Z}/\ell\mathbb{Z}$. For example take $\ell = 2$, then $\mu_2 = \{1, -1\} \cong \mathbb{Z}/2\mathbb{Z}$, but this does not hold when the characteristic of K equals 2 because then $1 \equiv -1$.

Definition (Weil pairing). The *Weil pairing* is defined with the ℓ-torsion and the ℓ^{th} roots of unity as

$$e_\ell : E[\ell] \times E[\ell] \to \mu_\ell.$$

Two elements of the ℓ-torsion are mapped to an ℓ^{th} root of unity.

To explicitly state how the Weil pairing is computed one must first know some concepts. The *divisor group of a curve E* is thr free abelian group generated by the points on E and is denoted by $Div(E)$. Elements of $Div(E)$ are called *divisors*, denoted by D, and can be written as a finite formal sum:

$$D = \sum_{P \in E} n_P(P),$$

where n_P is an integer and only finitely many n_P's are not equal to zero. Pay attention, $n_P(P)$ is not the same as $[n_P]P$, the first is an element of the formal sum of a divisor and the second notation is the addition law of the elliptic curve group. The *degree of D* is defined by $deg(D) = \sum_{P \in E} n_P$.

Given a function f on E, a degree 0 divisor $div(f)$ can be built from the zeros and poles of f, simply by forming a formal sum.

$$div(f) = \sum_{P \in E} ord_P(f)(P) = \sum \text{zeros of } f - \sum \text{poles of } f.$$

One finds the multiplicity of f at P, when f has a zero at P one takes $ord_P(f)$ as positive multiplicity and $ord_P(f)$ is taken minus the multiplicity when f has a pole at P. When P is not a zero and not a pole of f, $ord_P(f)$ is 0. Together with the following proposition we get closer to defining the Weil pairing explicitly.

Proposition 6.1. *$\sum P = \mathcal{O}$ if and only if there exists a function g on E with $div(g) = \sum \big((P)\big) - \delta(\mathcal{O})$.*

Here the first summation is the additive group law on E and the second is the formal sum. The δ equals the number of points on E that are summed.

One can also evaluate a function f at a divisor $D = \sum_{P \in E} n_P(P)$:

$$f(D) = \prod f(P)^{n_P}.$$

Computing the Weil pairing for two points P and Q, is done as follows. First check if P and Q both have order ℓ, otherwise it does not make sense. Now one needs to compute $f_P(D_Q)$ where f_P is a function such that $div(f_P) = \ell(P) - \ell(\mathcal{O})$ and D_Q denotes a suitable divisor from the divisor class of $(Q) - (\mathcal{O})$. According to the proposition above such a function f_P exists. (Remember P has an order that divides ℓ.) The divisor class of $(Q) - (\mathcal{O})$ is defined as all the divisors that can be written as $(Q) - (\mathcal{O}) + div(h)$ for a function h. In [Joux, 2002] is was stated that some choices of D_Q are not suitable but $D_Q = (Q \bigoplus R) - (R)$ is a good choice, where R is a randomly selected point on E.

To clarify the proposition and the Weil pairing, a short explanation why $(Q \bigoplus R) - (R)$ is in the divisor class of $(Q) - (\mathcal{O})$, is now given. The question is whether such a function h exists. The function h is in this case the quotient of the vertical line ℓ_1 through $Q \bigoplus R$ and the line ℓ_2 through R and Q. Note that ℓ_1 and ℓ_2 intersect at $-(Q \bigoplus R)$ as a result of the addition law on E. Furthermore, the equality $div(\ell_1/\ell_2) = div(\ell_1) - div(\ell_2)$ holds. Now we compute the seperate divisors using the proposition.

The zeros of ℓ_1 are $Q \bigoplus R$ and $-(Q \bigoplus R)$. Together with the fact that $Q \bigoplus R \bigoplus -(Q \bigoplus R) = \mathcal{O}$ holds and the proposition, we know

$$div(\ell_1) = (Q \oplus R) + (-(Q \oplus R)) - 2(\mathcal{O}).$$

And similarly,

$$div(\ell_2) = (R) + (Q) + (-(Q \oplus R)) - 3(\mathcal{O}).$$

As a result

$$div(\ell_1/\ell_2) = (Q \oplus R) + (-(Q \oplus R)) - 2(\mathcal{O}) - [(R) + (Q) + (-(Q \oplus R)) - 3(\mathcal{O})]$$

$$= (Q \oplus R) + (\mathcal{O}) - (R) - (Q).$$

So indeed $(Q \bigoplus R) - (R)$ can be written in the form $(Q) - (\mathcal{O}) + div(h)$:

$$(Q) - (\mathcal{O}) + div(\ell_1/\ell_2) = (Q) - (\mathcal{O}) + (Q \oplus R) + (\mathcal{O}) - (R) - (Q)$$

$$= (Q \oplus R) - (R).$$

Finally we can state the Weil pairing completely. Assume E is an elliptic curve over $\mathbb{F}_q$, with $q = p^n$ and p a prime. Take an ℓ that divides $p^{rn} - 1$ for a reasonably small r and pick the two ℓ-torsion points P and Q. The Weil pairing is:

$$e_\ell(P, Q) = f_P(D_Q)/f_Q(D_P).$$

Here f_Q and D_P have the same meaning as described above for f_P and D_Q; f_Q is a function such that $div(f_Q) = \ell(Q) - \ell(\mathcal{O})$ and D_P represents the divisor class of $(P) - (\mathcal{O})$. The condition for ℓ will not be further discussed in this section.

6.3 Approach to a solution

Now the extension protocols and the concept of pairings are bridged. How are the extension protocols implemented? Unfortunately, some more theory needs to be explained to obtain all the tools we need. This theory will be explained in the next section. The second section will be used to set up the pairing we want to use to implement the extension protocols. Thereafter an example is given where this alternative pairing is implemented and computed. Next to that, two other attempts to implement the alternative pairing are demonstrated. In the final subsection the extension protocols are implemented with the first example.

6.3.1 More theory

Definition (Embedding degree). Let E be an elliptic curve over a finite field $\mathbb{F}_p$ and let $n|\#E(\mathbb{F}_p)$ such that $gcd(n,p) = 1$. The *embedding degree* $k(p,n)$ is the smallest positive integer such that $n|(p^{k(p,n)} - 1)$.

Then the n^{th} roots of unity $\mu_n \subseteq \mathbb{F}_{p^{k(p,n)}}$ [Galbraith, 2012]. When n is taken equal to $\#E(\mathbb{F}_p)$, the embedding degree is denoted as k.

For example, take a look at the example of chapter 3. The elliptic curve $E : y^2 = x^3 - 3x + 3$ was considered over $\mathbb{F}_7$ and $\#E(\mathbb{F}_7) = 10$. The embedding degree $k(7,5)$ is not 1 $(5 \nmid (7-1) = 6)$, not 2 $(5 \nmid (7^2-1) = 48)$, not 3 $(5 \nmid (7^3-1) = 342)$, but 4 $(5 \mid (7^4-1) = 2400)$.

Let it be clear that if one wants to use an embedding degree in cryptography, it should be sufficiently small. Otherwise, $\mathbb{F}_{p^{k(p,n)}}$ will too large to decently perform computations. However, to maintain the security of the system k should not be too small, see Chapter 3.

Definition (Twist). Let E be an elliptic curve over the field K. A *twist* of $E(K)$ is an elliptic curve E' over K such that $E' \cong E$, over the algebraic closure $\bar{K}$ of K.

The algebraic closure of a field is the smallest extension of this field that is algebraically closed. In this thesis the general form of the elliptic curve we are considering is

$$E : y^2 = x^3 + ax + b.$$

The general form of a twist of E then becomes

$$E' : y^2 = x^3 + au^4x + bu^6, \qquad \text{with } u \in \bar{\mathbb{F}}_p \text{ such that } au^4, bu^6 \in \mathbb{F}_p.$$

The isomorphism that transforms a point in $E'(\bar{K})$ to a point in $E(\bar{K})$ is defined as

$$\phi : (x', y') \mapsto \left(\frac{x'}{u^2}, \frac{y'}{u^3}\right).$$

And the other way around

$$\phi^{-1} : (x, y) \mapsto (u^2x, u^3y).$$

One might question the use of twists of elliptic curves. The reason is simple. It is a perfect way to create two elliptic curve groups with the same number of elements,

because of the isomorphic property. As a result of au^4 and bu^6 being contained in $\mathbb{F}_p$, one does not have to work in huge extensions.

Look at the following example and one can see the relation between the embedding degree and $\bar{K}$.

Example. Let $E : y^2 = x^3 + 72$ defined over $\mathbb{F}_{p=103}$ then $\#E(\mathbb{F}_{103}) = 84$ and take $n = 7$. The embedding degree $k(103, 7) = 6$. The twists of E then are of the form $E' : y^2 = x^3 + 72u^6$. [Costello, 2012] The u should be an element chosen from $\mathbb{F}_{p^{k(103,7)}} \backslash \mathbb{F}_p$. In this case a u for which $u^6 + 2 = 0$ holds is a good choice. So a twist E' is the curve $y^2 = x^3 - 144$ over $\mathbb{F}_{103^6}$. A little check: $\#E(\mathbb{F}_{103^6}) = \#E'(\mathbb{F}_{103^6}) = 1194051169584$.

The conclusion is: when one considers an elliptic curve group $E(\mathbb{F}_p)$ with p a prime, the embedding degree will reveal the field extension such that $E(\mathbb{F}_{p^{k(p,n)}}) \cong E'(\mathbb{F}_{p^{k(p,n)}})$.

There is more. One can ascribe a degree to a twist. This has everything to do with the $u \in \mathbb{F}_{p^{k(p,n)}} \setminus \mathbb{F}_p$. One can find this by looking for a u for which $u^d + \rho = 0$ holds, with $\rho \in \mathbb{F}_p^*$. There are four options for degree d.

- *d=2 quadratic twist.* An elliptic curve group of $E : y^2 = x^3 + ax + b$ over a finite field $\mathbb{F}_p$ can always have a degree 2 twist $E' : y^2 = x^3 + au^4x + bu^6$.
- *d=3 cubic twist.* When $a = 0$ the elliptic curve group $E : y^2 = x^3 + b$ over a finite field $\mathbb{F}_p$ can have a twist $E' : y^2 = x^3 + bu^6$ of degree 3.
- *d=4 quartic twist.* When the elliptic curve has no constant factor ($b = 0$) the elliptic curve group $E : y^2 = x^3 + ax$ over a finite field $\mathbb{F}_p$ can have a degree 4 twist $E' : y^2 = x^3 + au^4x$.
- *d=6 sextic twist.* An elliptic curve group $E : y^2 = x^3 + b$ over a finite field $\mathbb{F}_p$ can also have a twist $E' : y^2 = x^3 + bu^6$ of degree 6.

For example, an elliptic curve group $E : y^2 = x^3 + 6x$ over a finite field will therefore never have a twist of degree 3. In the example a twist of degree 6 was given. The most popular are twist of degree 4 or 6 [Costello, 2012].

Definition (Isogeny). Let E_1 and E_2 be two elliptic curves over K. An *isogeny* is a morphism

$$\phi : E_1 \longrightarrow E_2, \qquad \text{satisfying } \phi(\mathcal{O}_{E_1}) = \mathcal{O}_{E_2}.$$

Two elliptic curves E_1 and E_2 are called *isogenous* if there exists an isogeny between E_1 and E_2. The *zero isogeny* is the constant map $\phi : E_1 \to E_2$ such that $\phi(P) = \mathcal{O}_{E_2}$ for all $P \in E_1(K)$. The points $\mathcal{O}_{E_1}$ and $\mathcal{O}_{E_2}$ are the points at infinity of E_1 and E_2, respectively.

Theorem 6.2. *Let $\phi : E_1 \to E_2$ be a morphism of the curves E_1 and E_2. Then ϕ is either constant or surjective. [Silverman, 2009]*

From theorem 6.2 is follows that an isogeny is either the zero isogeny or is a finite map of curves: $\phi(E_1) = E_2$.

Theorem 6.3. *Let E_1 and E_2 be elliptic curves over K and let $\phi : E_1 \to E_2$ be a non-zero isogeny of degree m (the degree of the morphism). Then there is an isogeny $\hat{\phi} : E_2 \to E_1$ such that*

$$\hat{\phi} \circ \phi : E_1 \longrightarrow E_1$$

which multiplies the elements of E_1 by $[m]$. [Galbraith, 2012]

Here $\hat{\phi}$ is called the *dual isogeny* of ϕ.

Theorem 6.4. *Two elliptic curves E_1 and E_2 over a finite field are isogenous if and only if they have the same number of points. [Tate, 1966]*

An isogeny of degree m is also known as an m-isogeny. One can construct a *ℓ-isogeny graph* of prime degree ℓ, where ℓ is not equal to the characteristic of the field over which the elliptic curves of the isogenies are defined. An ℓ-isogeny graph consists of nodes en $\ell + 1$ vertices from each node. The nodes represent elliptic curves and the vertices represent ℓ-isogenies. Two connected elliptic curves form a so called *volcano*.

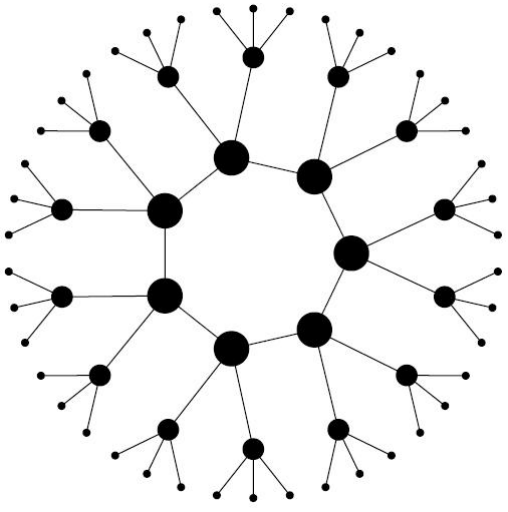

FIGURE 6.1: A 3-isogeny graph. [De Feo et al., 2012]

Definition (Supersingular and ordinary curves). An elliptic curve E is called *supersingular* when $E(\bar{K})[p^r] \cong \{0\}$, where p is the characteristic of field K, $r = 1, 2, 3, \ldots$ and

$\bar{K}$ is again the algebraic closure of K. If E is not supersingular it is called *ordinary* and $E(\bar{K})[p^r] \cong \mathbb{Z}/p^r\mathbb{Z}$.

When two elliptic curves are isogenous, they are both ordinary or both supersingular [Sutherland, 2012]. All supersingular elliptic curves in an isogeny graph can easily be obtained, i.e. it is easy to find these isogenies and therefore easy to find the elliptic curves.

In order to classify pairings that use points of elliptic curve groups as input one needs to look at how efficiently an isogeny graph can be constructed.

6.3.2 Alternative pairing

As Weitenberg proclaimed the pairing included in the extension protocols needs to be a type III pairing. Remember, this means there does not exist an efficiently computable homomorphism between the two groups responsible for the input of the pairing. The Weil pairing uses two points from the same elliptic curve group. Therefore this pairing itself is not suitable for our protocols, because the identity isogeny is efficiently computable. This is also a reason why supersingular curves should be avoided, because there isogeny graphs are easily found.

Therefore a new pairing is unavoidable though the Weil pairing is still of much use. The alternative pairing is a modification of the Weil pairing.

Definition (Distortion map). Let E be an elliptic curve over $\mathbb{F}_p$, let $n|\#E(\mathbb{F}_p)$ be prime, $e : E[n] \times E[n] \to \mu_n$ be an non-degenerate and bilinear pairing and let $P \in E(\mathbb{F}_p)[n]$. A *distortion map* with respect to E, n, e and P is an endomorphism $\psi : E \to E$ such that $e(P, \psi(P)) \neq 1$.

This is similar to what happens in the alternative pairing. However ϕ is not an endomorphism but an isomorphism from $E_2[q]$ to $E_1[q]$ both defined over $\mathbb{F}_{p^k}$, with p and q prime. In fact ϕ is an isogeny from E_2 to E_1. The alternative pairing e_{alt} is thus of the form

$$e_{alt} : E_1[q] \times E_2[q] \to \mu_q.$$

And explicitely it is defined as

$$e_{alt}(P, Q) = e_\ell(P, \phi(Q)).$$

In figure 6.2 the setup of the alternative pairing is illustrated. This alternative pairing is bilinear, anti-symmetric and non-degenerate because the Weil pairing has these properties. Moreover, $\phi\,|_{q-torsion}$ is an isomorphism, because the degree of ϕ is coprime to q.

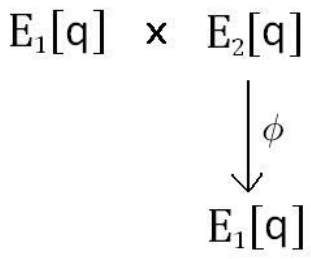

FIGURE 6.2: The alternative pairing.

Since ϕ maps points from the q-torsion of E_2 to the q-torsion of E_1 only, this is sufficient to know.

6.3.3 Example with $y^2 = x^3 + b$

The concept of the alternative pairing might still be a bit impalpable. Though some aspects must be clear. Point Q can not be a multiple of point P, because then $e_{alt}(P, Q) = e_{alt}(P, aP) = e_{alt}(P, P)^a = 1^a = 1$. This is not a safe choice. However point Q must, like point P, have prime order q; this is how the alternative pairing works, one takes two points from the q-torsion. Furthermore, we need to find two elliptic curves that are isogenous to each other and have the properties given in the description of the extension protocols. The order of the elliptic curve group E_1 should be prime and the elliptic curve group E_2 should have the same order, otherwise they are not isogenous. The alternative pairing is computed for an example.

As a recommendation: appendix I should be placed aside this text to completely understand the procedure from the stage where the primes p and q are chosen. The main idea of how to find the needed curves and points is taken from the handbook of Magma, though some changes were needed to make it usable for this problem.

Take the elliptic curve $y^2 = x^3 + b$, with $b \neq 0$. An elliptic curve group with a prime order and the point P needs to be found. One way to do this, is checking all the options of the order of E_1 and pick one that can be prime; my supervisor came with this idea.

When one has an elliptic curve $E_1(\mathbb{F}_p)$ and $p \equiv 1 \bmod 6$, p can be written as $N(\eta + \xi u)$ for certain $\eta, \xi \in \mathbb{Z}$ and $u = \frac{1}{2} + \frac{1}{2}\sqrt{-3} \in \mathbb{C}$. The order of u in $\mathbb{C}^*$ is 6. $N(\eta + \xi u)$ is the norm of $\eta + \xi u$ and is equal to

$$N(\eta + \xi u) = (\eta + \xi u)(\overline{\eta + \xi u}) = \eta^2 + \eta\xi(u + \bar{u}) + \xi^2 = \eta^2 + \eta\xi + \xi^2.$$

The *trace map* is defined as:

$$trace : \mathbb{Z}[u] \longrightarrow \mathbb{Z},$$

$$\alpha \longmapsto \alpha + \bar{\alpha}.$$

For clarity, $\mathbb{Z}[u]$ is here defined as $\mathbb{Z}[X]/(X^2 - X + 1) = \{\eta + \xi u | \eta, \xi \in \mathbb{Z}\}$. The order, $\#E_1(\mathbb{F}_p)$, is one of the following:

- $p + 1 - trace(\eta + \xi u) = p + 1 - 2\eta - \xi$.
- $p + 1 - trace(u(\eta + \xi u)) = p + 1 - \eta + \xi$.
- $p + 1 - trace(u^2(\eta + \xi u)) = p + 1 + \eta + 2\xi$.
- $p + 1 - trace(u^3(\eta + \xi u)) = p + 1 + 2\eta + \xi$.
- $p + 1 - trace(u^4(\eta + \xi u)) = p + 1 + \eta - \xi$.
- $p + 1 - trace(u^5(\eta + \xi u)) = p + 1 - \eta - 2\xi$.

Moreover, for each of the six possibilities there is an $a \in \mathbb{F}_p^*$ such that $y^2 = x^3 + a$ has the number of points listed above.

In the example from the Magma handbook η is chosen $6s^2 + 2s + 1$ and ξ is taken $2s$, these are polynomials. When all options for the order of $E_1(\mathbb{F}_p)$ are calculated, one can see that only the second and sixth option can be prime. The others are all divisible by some integer. Again the example from the handbook is followed; $q = \#E_1(\mathbb{F}_p)$ is taken to be $p + 1 - \eta + \xi$. One can calculate p and q;

$$p = 36s^4 + 36s^3 + 24s^2 + 6s + 1,$$

$$q = 36s^4 + 36s^3 + 18s^2 + 6s + 1.$$

This is how the polynomials ps and qs in appendix I are obtained. Note that $p \equiv 1 \bmod 6$ holds. The value of z is a start value. The program iterates over z and stops until it finds one such that ps and qs are both prime. The program command states `IsProbablePrime`, but for small numbers (and these are) probable prime means prime and this is faster than checking whether it is prime.

Once the right p and q are found the finite field $\mathbb{F}_p$ is created. Thereafter the smallest b is found such that $b + 1$ is a square and $G_1 = (1, y)$ is a generator of $E_1(\mathbb{F}_p)$ and has order q. Here y is chosen as the square root of $b + 1$. The elliptic curve group $E_1(\mathbb{F}_p)$ is now found, because b has a fixed value; $E_1 : y^2 = x^3 + 18$ for

$$p = 24978577110957807134030566063991512750990207247 23.$$

Then as a verification there is a check whether E_1 has indeed order q. The embedding degree k is equal to 12. This is not true in general but for the cases considered here. When one would change the start value z the embedding degree would also be 12. The embedding degree is also checked and the extension field $\mathbb{F}_{p^{12}}$ is created. The first entry of the alternative pairing is point P. This is point G_1 taken as an element of $E_1(\mathbb{F}_{p^{12}})$. The order of point P is indeed a prime like the extension protocol requires, it is q.

The next step is finding the second elliptic curve E_2 isogenous to E_1. This can be done by finding a twist of E_1. The embedding degree is already known, so one needs to find an elliptic curve such that $E_1(\mathbb{F}_{p^{12}}) \cong E_2(\mathbb{F}_{p^{12}})$. It is found in a similar way as E_1, only now the next b is found such that $b+1$ is a square and $G_2 = (1, y)$ is a generator of $E_2(\mathbb{F}_{p^{12}})$ and has order q. The curve that is found is $E_2 : y^2 = x^3 + 28$. After finding the right elliptic curve, the first elliptic curve is lifted from $\mathbb{F}_p$ to $\mathbb{F}_{p^{12}}$. Then it is checked whether $E_1(\mathbb{F}_{p^{12}})$ and $E_2(\mathbb{F}_{p^{12}})$ are isomorphic and the isomorphism ϕ`isomorphism` is turned into the isogeny ϕ. The isogeny ϕ is

$$\phi : E_2 \longrightarrow E_1,$$

$$\phi : (x, y) \mapsto (209380928986641133276824737016007269173954900510 0x,$$

$$230028823044745242289735647565631463842808075374 6y).$$

With the resultant the number of points N of the elliptic curve group $E_2(\mathbb{F}_{p^{12}})$, and therefore also $\#E_1(\mathbb{F}_{p^{12}})$, is computed. Then the second entry, point $Q \in E_2(\mathbb{F}_{p^{12}})$, of the alternative pairing is created. Also Q must have order q. Factoring shows that q is twice a prime factor of N. Therefore Q is defined as a random point times the factors of N, excluding q^2. The second-last step is defining P as an element of $E_1(\mathbb{F}_{p^{12}})$, remember the input elements of the Weil pairing need to come from the same elliptic curve group. The alternative pairing can be computed with the Weil pairing using the isogeny ϕ;

$$e_{alt}(P, Q) = \texttt{WeilPairing(P,phi(Q),q)}.$$

6.3.4 Attempt of an example with $y^2 = x^3 + ax$

The alternative pairing was implemented with E_1 and E_2 of the form $y^2 = x^3 + b$. A similar method is used in an attempt to implement the alternative pairing with elliptic curves of the form $y^2 = x^3 + ax$, with $a \neq 0$.

Beforehand a remark must be made. Elliptic curve groups of this form always contain a point of order 2, namely the point $(0, 0)$. The tangent line at this point is always vertical.

The elliptic curve group will therefore never have a prime order. The suggestion is to find an elliptic curve of this form that contains $2q$ points, with q a prime.

Again the norm can be used to find all the possible orders of the elliptic curve group $E_1(\mathbb{F}_p)$. In this case $u = i$ and p can be written as $N(\eta + \xi u)$ for certain $\eta, \xi \in \mathbb{Z}$.

$$N(\eta + \xi i) = (\eta + \xi i)(\eta - \xi i) = \eta^2 + \xi^2.$$

Therefore p can be written as the sum of two squares. The different possibilities for the order of the group are again discovered with the trace map. To be precise, $\mathbb{Z}[i]$ is here defined as $\mathbb{Z}/(X^2 + 1) = \{\eta + \xi i | \eta, \xi \in \mathbb{Z}\}$. In this case there are four possibilities for $\#E_1(\mathbb{F}_p)$:

- $p + 1 - trace(\eta + \xi i) = p + 1 - 2\eta$.
- $p + 1 - trace(i(\eta + \xi i)) = p + 1 + 2\xi$.
- $p + 1 - trace(i^2(\eta + \xi i)) = p + 1 + 2\eta$.
- $p + 1 - trace(i^3(\eta + \xi i)) = p + 1 - 2\xi$.

After a lot of attempts (including writing several programs) to find suitable η and ξ such that $\#E_1(\mathbb{F}_p)$ is a prime times 2 and the embedding degree k is small, I merely managed to find an example with a p that is too small. It appears to be extremely difficult to find such an example with a p that is big enough. The example I found, where $p = 13 = 2^2 + 3^2$, is the elliptic curve group of $y^2 = x^3 + 2x$ over $\mathbb{F}_{13}$. The number of points contained in the group is 10, the embedding degree is 4 and the generator $G_1 = (1, 9)$.

If one would try to implement the alternative pairing with this example it would not be a realistic approach to the problem. Therefore, this example is left as it is and not further elaborated.

6.3.5 Attempt of an example with $y^2 = x^3 + x + b$

Another attempt was made to find an example of a curve of the form $y^2 = x^3 + x + b$ which satisfies the needed properties. In this case the number of points in the elliptic curve group can be prime and this is the goal.

The trick with the trace map does not work here. Therefore, I wrote different programs to find such an example, but I could not find better examples than the following two.

The elliptic curve group of $y^2 = x^3 + x + 13$ over $\mathbb{F}_{59}$ is an example, although the embedding degree $k = 44$ is quite large. The number of points contained in the group is 67 and the generator is $G_1 = (1, 29)$.

Another, better, example is the elliptic curve group $y^2 = x^3 + x + 3$ over $\mathbb{F}_{31}$. The embedding degree $k = 10$ is smaller and has now an acceptable value. The number of points contained in the group is 41 and the generator is $G_1 = (1, 25)$.

The main problem was to find a curve with a low embedding degree and a large prime p. With the time in mind, I decided not to investigate this type of elliptic curve further. This decision is supported by the following theorem.

Theorem 6.5. *Let p and E be randomly chosen, with a prime in the interval $M/2 \leq p \leq M$ and an elliptic curve defined over $\mathbb{F}_p$ having a prime number q of points. The probability that $q|(p^k - 1)$ for some $k \leq (\log p)^2$ is less than*

$$c\frac{(\log M)^9(\log\log M)^2}{M}$$

for an effectively computable positive constant c. [Balasubramanian & Koblitz, 1998]

6.3.6 Implementing the extension protocols

As mentioned before, the implementation of the extension protocols will be executed with the example given in subsection 6.3.3. The whole protocols are implemented except for the proofs of knowledge over d in the Issuing protocol and over σ and $\tilde{r}$ in the Showing protocol. The reason for this is simple the lack of time. The implementations can be viewed in appendices J and K. Both implementations follow the protocols in section 6.1 quite well, though there are some aspects that do not and need further explanation.

The implemention of the extended Issuing protocol is indistinct due to several aspects. The most easy aspect is that certain characters in the protocol are already used in the implementation; they need a new "name". In the protocol the attribute values are denoted by k_0 and k_i, though k is already the character of the embedding degree. To avoid confusion and to make it easier to understand where the characters stand for, the attributes values are denoted by `av0` and the vector `av`. Furthermore, there is another element that is called k in the protocol. In the implementation this element is denoted by `ka`.

To make sure the inverse of `(gpowerlambdaasinteger -1) div n` exists, g is chosen such that this criterion holds.

The remarkable part of the implementation lies in the definition of `av0bar` and `d`. These elements have their bases in $\mathbb{Z}/n\mathbb{Z}$ and most of their exponents in $\mathbb{F}_q$. In order to make the calculations meaningful, the exponents need to be lifted to their representatives in $\mathbb{Z}/n\mathbb{Z}$. This is done by choosing p_2 to be q, as Weitenberg suggested. Now n is a multiple of q. The question may rise whether the security of the Paillier encryption is harmed, because in the encryption n is allowed to be public but p_1 and p_2 are not. The prime q is public and therefore p_2 is public. With the broadcasting of p_2, p_1 is becomes public. The answer to the question is 'yes', though this does not harm the extension of the Issuing protocol. The reason for this is that this encryption is only used as a tool within the protocol, but it does not depend on it. Now the lifting has become a formality, because n is bigger than q. When one lifts an element from $\mathbb{F}_q$ to $\mathbb{Z}/n\mathbb{Z}$, the representatives are the same. The exponent `beta*av0` of `av0bar` is lifted by simply adding `mod n` to it. The exponents `z`, `x0`, `ka` and `gamma` are first lifted to $\mathbb{Z}/n\mathbb{Z}$. After that their representative in $\mathbb{Z}$ is found. The last step is executed in order to make it computable for Magma.

The question might rise whether it is possible to choose p_1 equal to 1. This is not a prime, as Paillier designed his encryption, but what will happen when n is equal to q? Can the protocols work in $\mathbb{Z}/q^2\mathbb{Z}$? It is possible. For a computable implementation λ should be equal to $p_2 - 1$ but the implementation of the complete U-prove extension runs. However, it remains unproven if this affects the security of the extension protocols.

It is rightfully mentioned in Weitenberg [2012] that 'it should be noted that 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.' This is due to the required size of n^2.

The extension of the Showing protocol has one small aspect that needs to be explained. The values of the r_i are denoted in a vector `rav` to avoid confusion with the `i` used in the for loop.

Magma is able to execute the extended Issuing protocol when p_1 is chosen around 2^{95}. The same holds for the extended Showing Protocol when p_1 is chosen around 2^{90}.

Chapter 7

Conclusion

The extension protocols of U-prove have been implemented successfully. Of course, there are some limitations and maybe the protocols can be implemented more efficiently. The Identity Provider and the Verifier, as in the original U-prove, do not know whether they communicated with the same User. Most importantly, hopefully the unlinkability of the Showing protocol to itself is achieved; the User should be able to use a credential multiple times. Nonetheless, it is not proven that the presumption, of the extended Showing protocol being unlinkable, is true.

Furthermore, the signature for the attribute commitment is still valid. However, the security of my implementations are questionable. As mentioned in Chapter 3, the embedding degree should be at least 21 to keep the system secure. This is not the case in the implementation of the extension protocols and since it was already difficult to find this example, it is very questionable whether these protocols are useful in practice. Moreover, the used prime numbers are quite small.

One motive for this research was to investigate whether the extension protocols are implementable and not only work on paper. The outcome is that the protocols are indeed implementable for the example that is found. The second motive was to examine the difficulty of implementing a pairing. This resulted in a lot of procedures to guarantee the system possesses all the right conditions. The conclusion drawn from this, is that it takes a lot of effort to create the right setting. After all, once the setting is obtained it does not take much time to compute the pairing. Pairings deliver a secure setting, but obtaining it is costly.

Many researchers are constantly trying to improve the security of cryptographic systems. Therefore, they are searching for methods to break the Discrete Logarithm Problem for very large numbers. The current record is discovered by Antoine Joux on 21 May 2013

in a finite field of characteristic 2. He and is team computed the discrete logarithms in the field with 2^{6168} elements. [Joux, 2013]

The same is done for the Elliptic Curve Discrete Logarithm Problem. In July 2009 a team of researcher managed to compute the discrete logarithm on an elliptic curve modulo a 112-bit field. It took a cluster of more than 200 PlayStation 3 game consoles almost six months to execute the calculations; it started on 13 January 2009 and finished on 8 July 2009. With the latest version of the code it takes three and a half months. The research team is very precise about the execution: 'In total, about 8.5 * 1016 elliptic curve point additions were performed, which translates to approximately 1018 modular additions and multiplications on 112-bit integers.'. [EPFL, 2013]

The development of cryptography grows extremely fast with the digital society we live in and will continue to grow for a long time.

Appendix A

ElGamal encryption

The Magma-code to implement the ElGamal encryption.

```
p := RandomPrime(299);
k := GF(p);
x := Random(p-1);
f := Factorization(p-1);
"The multiplicative group G we use is the", k,
 "without the zero-element, of course.";

g := PrimitiveElement(k);
"Generator=", g;

h := g^x;
"h=", h;

m := Random(p-1);
while m eq 0 do
  m:= Random(p-1);
end while;

y := Random(p-1);
c1 := g^y;
c2 := m*h^y;
"c1=", c1;
"c2=", c2;
```

```
decoding := c2/(c1^x);
"decoding=", decoding;
"m=", m;
if m eq decoding then
  "decoding successful";
else
  "decoding failed";
end if;
```

Appendix B

Paillier encryption

The Magma-code to implement the Paillier encryption.

```
p1 := RandomPrime(109);
p2 := RandomPrime(109);
n := p1*p2;
Z := Integers();
nZ2 := ideal<Z | n^2>;
Zn2 := quo<Z|nZ2>;
g := Random(Zn2);

while Gcd(Order(g),n) ne n or Gcd(g,n^2) ne 1 do
   g := Random(Zn2);
end while;

"n=", n;
"g=", g;

m := Random(n-1);
rho := Random(n-1);
while Gcd(rho,n) ne 1 do
   rho := Random(n-1);
end while;
rhotilde := Zn2!rho;

c := (g^m)*(rhotilde^n);
```

```
lambda := Lcm(p1-1,p2-1);
cpowerlambdaasinteger := Z!(c^lambda);
gpowerlambdaasinteger := Z!(g^lambda);
inversedenominator := Modinv((gpowerlambdaasinteger -1) div n,n);

decoding := (((cpowerlambdaasinteger - 1) div n)*inversedenominator)
   mod n;
decodingasinteger := Z!decoding;

"decoding=", decoding;
"m=", m;
if m eq decodingasinteger then
  "decoding successful";
else
  "decoding failed";
end if;
```

Appendix C

ElGamal signature scheme

The Magma-code to implement the ElGamal signature scheme.

```
n := 396;
p := RandomPrime(n);
while p eq 2 do
   p := RandomPrime(n);
end while;
"p=", p;
k := GF(p);
x := Random(p-1);
"x=", x;
Z := Integers();

g := PrimitiveElement(k);
"Generator=", g;

h := g^x;
"h=", h;

m := Random(p-1);

y := Random(p-2);
while y eq 0 or Gcd(y,p-1) ne 1 do
  y := Random(p-2);
end while;
"y=", y;
```

```
inversey := Modinv(y, p-1);
"inversey=", inversey;

s1 := g^y;
s1asinteger := Z!s1;
s1asintegermodp := s1asinteger mod p;

s2 := ((Hash(m)-x*s1asintegermodp)*inversey) mod (p-1);

while s2 eq 0 do
   y := Random(p-2);
   while y eq 0 or Gcd(y,p-1) ne 1 do
     y := Random(p-2);
   end while;
   inversey := Modinv(y, p-1);
   s1 := g^y;
   s1asinteger := Z!s1;
   s1asintegermodp := s1asinteger mod p;
   s2 := ((Hash(m)-x*s1asintegermodp)*inversey) mod (p-1);
end while;

"s1=", s1;
"s2=", s2;
"Hash(m)=", Hash(m);
"g^Hash(m)=", g^Hash(m);
"h^s1*s1^s2=", (h^s1asintegermodp)*(s1^s2);

if 0 lt s1asintegermodp and s1asintegermodp lt p and 0 lt s2 and
s2 lt (p-1) and g^Hash(m) eq (h^s1asintegermodp)*(s1^s2) then
   print "Valid signature";
else
   print "Invalid signature";
end if;
```

Appendix D

Schnorr proof of knowledge

The Magma-code to implement the Schnorr's proof of knowledge protocol.

```
p := RandomPrime(191);
a := Random(p-1);
b := Random(p-1);
while -16*(4*a^3+27*b^2) eq 0 do
   a := Random(p-1);
   b := Random(p-1);
end while;
E := EllipticCurve ([GF(p) | a , b ]);

while IsPrime(#E) eq false do
   p := RandomPrime(191);
   a := Random(p-1);
   b := Random(p-1);
   while -16*(4*a^3+27*b^2) eq 0 do
      a := Random(p-1);
      b := Random(p-1);
   end while;
E := EllipticCurve ([GF(p) | a , b ]);
end while;
"p=", p;
"Elliptic curve we use=", E;
"Group over this elliptic curve=", AbelianGroup(E);
q := #E;
```

```
P := Random(E);
while Order(P) ne q do
   P := Random(E);
end while;
"P=", P;

x := Random(q-1);
X := x*P;
"X=", X;

w := Random(q-1);
W := w*P;
"W=", W;

gamma := Random(q-1);
"gamma=", gamma;

r := (gamma*x+w) mod q;
"r=", r;

if W eq (r*P-gamma*X) then
   "Proof of knowledge";
else
   "No proof of knowledge";
end if;
```

Appendix E

Schnorr signature scheme

The Magma-code to implement the Schnorr signature scheme.

```
p := RandomPrime(182);
a := Random(p-1);
b := Random(p-1);
while -16*(4*a^3+27*b^2) eq 0 do
   a := Random(p-1);
   b := Random(p-1);
end while;
E := EllipticCurve ([GF(p) | a , b ]);
m := 0100110010110;  %This is an example.

while IsPrime(#E) eq false do
   p := RandomPrime(182);
   a := Random(p-1);
   b := Random(p-1);
   while -16*(4*a^3+27*b^2) eq 0 do
      a := Random(p-1);
      b := Random(p-1);
   end while;
E := EllipticCurve ([GF(p) | a , b ]);
end while;
"p=", p;
"Elliptic curve we use=", E;
"Group over this elliptic curve=", AbelianGroup(E);
q := #E;
```

```
P := Random(E);
while Order(P) ne q do
   P := Random(E);
end while;
"P=", P;

x := Random(q-1);
X := x*P;
"X=", X;

w := Random(q-1);
W := w*P;

m := IntegerToString(m);
HashW:= Hash(W);
hashWstring := IntegerToString(HashW);
mcathashW := m cat hashWstring;

gamma := Hash(mcathashW);
"gamma=", gamma;

r := (gamma*x+w) mod q;
"r=", r;

"Signature is (",gamma,",",r,")";

rPgammaX := r*P-gamma*X;
HashrPgammaX := Hash(r*P-gamma*X);
hashrPgammaxstring := IntegerToString(HashrPgammaX);
mcathashrPgammaX := m cat hashrPgammaxstring;

if gamma eq Hash(mcathashrPgammaX) then
   "Signature correct";
else
   "Signature incorrect";
end if;
m := IntegerToString(m);
HashW:= Hash(W);
```

```
hashWstring := IntegerToString(HashW);
mcathashW := m cat hashWstring;

gamma := Hash(mcathashW);
"gamma=", gamma;

r := (gamma*x+w) mod q;
"r=", r;

"Signature is (",gamma,",",r,")";

rPgammaX := r*P-gamma*X;
HashrPgammaX := Hash(r*P-gamma*X);
hashrPgammaxstring := IntegerToString(HashrPgammaX);
mcathashrPgammaX := m cat hashrPgammaxstring;

if gamma eq Hash(mcathashrPgammaX) then
   "Signature correct";
else
   "Signature incorrect";
end if;
```

Appendix F

Schnorr blind signature scheme

The Magma-code to implement the Schnorr blind signature scheme.

```
p := RandomPrime(172);
a := Random(p-1);
b := Random(p-1);
while -16*(4*a^3+27*b^2) eq 0 do
   a := Random(p-1);
   b := Random(p-1);
end while;
E := EllipticCurve ([GF(p) | a , b ]);
m := 0100110010110; % This is an example.

while IsPrime(#E) eq false do
   p := RandomPrime(172);
   a := Random(p-1);
   b := Random(p-1);
   while -16*(4*a^3+27*b^2) eq 0 do
      a := Random(p-1);
      b := Random(p-1);
   end while;
E := EllipticCurve ([GF(p) | a , b ]);
end while;
"p=", p;
"Elliptic curve we use=", E;
"Group over this elliptic curve=", AbelianGroup(E);
q := #E;
```

```
P := Random(E);
while Order(P) ne q do
   P := Random(E);
end while;
"P=", P;

x := Random(q-1);
X := x*P;
"X=", X;

w := Random(q-1);
W := w*P;

alpha := Random(q-1);
beta := Random(q-1);
Wtilde := W+alpha*X+beta*P;

m := IntegerToString(m);
HashWtilde:= Hash(Wtilde);
hashWtildestring := IntegerToString(HashWtilde);
mcathashWtilde := m cat hashWtildestring;


gamma := Hash(mcathashWtilde);
gammatilde := (gamma+alpha) mod q;
"gammatilde=", gammatilde;

r := (gammatilde*x+w) mod q;

if W eq (r*P-gammatilde*X) then
   "Signature correct";
else
   "Signature incorrect";
end if;

rtilde := (r+beta) mod q;

"Signature is (",gamma,",",rtilde,")";
```

Appendix G

U-prove protocol: Issuing protocol

The Magma-code to implement the Issuing protocol of U-prove.

```
p := RandomPrime(339);
a := Random(p-1);
b := Random(p-1);
while -16*(4*a^3+27*b^2) eq 0 do
   a := Random(p-1);
   b := Random(p-1);
end while;
E := EllipticCurve ([GF(p) | a , b ]);

while Gcd(#E,p-1) ne 1 do
   p := RandomPrime(339);
   a := Random(p-1);
   b := Random(p-1);
   while -16*(4*a^3+27*b^2) eq 0 do
      a := Random(p-1);
      b := Random(p-1);
   end while;
E := EllipticCurve ([GF(p) | a , b ]);
end while;
"p=", p;
"Elliptic curve we use=", E;
```

```
"Group over this elliptic curve=", AbelianGroup(E);

P := Random(E);
while Order(P) ne #E do
   P := Random(E);
end while;
"P=", P;
q := Order(P);

m := 10;
k := Matrix( 1, m, [1..m by 1]);
for i in [1..m] do
   k[1,i] := i;
end for;

x := Matrix( 1, m, [1..m by 1]);
for i in [1..m] do
   x[1,i] := Random(q-1);
   while Gcd(x[1,i],q) ne 1 do
      x[1,i] := Random(q-1);
   end while;
end for;
y := Random(q-1);
while Gcd(y,q) ne 1 do
   y := Random(q-1);
end while;

Y := y*P;
X := [ i*P : i in [1..m] ];
for i in [1..m] do
X[i] := x[1,i]*P;
end for;

w := Random(q-1);
W := w*P;
"W=", W;

k0 := Random(q-1);
alpha := Random(q-1);
```

```
beta := Random(q-1);
C := k0*P+ &+[ k[1,i]*X[i] : i in [1..m] ];
Wtilde := W+alpha*(Y+ &+[ k[1,i]*X[i] : i in [1..m] ])+beta*P;

HashC := Hash(C);
hashCstring := IntegerToString(HashC);
HashWtilde:= Hash(Wtilde);
hashWtildestring := IntegerToString(HashWtilde);
hashCcathashWtilde := hashCstring cat hashWtildestring;

gamma := Hash(hashCcathashWtilde);
gammatilde := gamma+alpha mod q;
"gammatilde=", gammatilde;
r := gammatilde*(y+ &+[ k[1,i]*x[1,i] : i in [1..m] ])+w;
"r=", r;

if W eq (r*P-gammatilde*(Y+ &+[ k[1,i]*X[i] : i in [1..m] ])) then
   "Signature valid";
   rtilde := r+beta+gamma*k0;
   rtildemodq := rtilde mod q;
   "Signature is (",gamma,",",rtildemodq,") ";
else
   "Signature invalid";
end if;
```

Appendix H

U-prove protocol: Showing protocol

The Magma-code to implement the Showing protocol of U-prove.

In this protocol we assume the Issuing protocol is run before this protocol. When the two protocols are run after each other, the prime p can be chosen between 0 and 2^{339}.

```
HashrtildePgammaCY := Hash(rtilde*P-gamma*(C+Y));
hashrtildePgammaCYstring := IntegerToString(HashrtildePgammaCY);
hashCcathashrtildePgammaCY := hashCstring cat hashrtildePgammaCYstring;

if gamma eq Hash(hashCcathashrtildePgammaCY) then
   "Signature validated by Verifier";
else
   "Signature found invalid by Verifier";
end if;

D := [ i : i in [1..m by 2] ];
"Which attribute has a public attribute value?: the X's with i=", D;

indexC := Matrix(  1, m-#D, [1..(m-#D) by 1]);
s := 1;
for i in [1..m] do
   v := i notin D;
   if v eq true then
      indexC[1,s] := i;
```

```
      s := s+1;
   end if;
end for;

t := [1..(m-#D) by 1];
for i in [1..(m-#D)] do
  t[i] := indexC[1,i];
end for;
IndexC := t;

wshowing := Matrix( 1, m+1, [1..(m+1) by 1]);
for i in [1..m+1] do
   wshowing[1,i] := Random(q-1);
end for;

What := wshowing[1,1]*P + &+[ wshowing[1,i+1]*X[i] : i in IndexC ];
"What=", What;

gammahat := Random(q-1);
while Gcd(gammahat,q) ne 1 do
   gammahat := Random(q-1);
end while;
gammahat;

r0 := gammahat*k0+wshowing[1,1];
"r0=", r0;
rshowingwhole := [ gammahat*k[1,i]+wshowing[1,i+1]: i in [1..m]];

CD := &+[ k[1,i]*X[i] : i in D ];
CC := C-CD;

if What+gammahat*CC eq r0*P+ &+[rshowingwhole[i]*X[i] : i in IndexC ]
   then
   "Proof of knowledge";
else
   "No proof of knowledge";
end if;
```

Appendix I

Alternative pairing $y^2 = x^3 + b$

The Magma-code to implement the alternative pairing for $y^2 = x^3 + b$.

```
Zs<s> := PolynomialRing(Integers());
ps := 36*s^4 + 36*s^3 + 24*s^2 + 6*s + 1;
qs := 36*s^4 + 36*s^3 + 18*s^2 + 6*s + 1;
z := 513235038556;  // some random start value
repeat
  z := z+1;
  p := Evaluate(ps, z);
  q := Evaluate(qs, z);
until IsProbablePrime(p) and IsProbablePrime(q);
p;
q;
Fp := FiniteField(p);
b := Fp!0;
repeat
  repeat b := b + 1; until IsSquare(b + 1);
  y := SquareRoot(b + 1);
  E1 := EllipticCurve([Fp!0, b]);
  G1 := E1![1, y];
until IsZero(q*G1);
E1;
#E1 eq q;  // just to verify
t := p + 1 - q;
t;
k := 12; // security multiplier
```

```
(p^k - 1) mod q; // check on p, q and k
Fpk := GF(p, k);
P := E1(Fpk)!G1;

repeat
  repeat b := b + 1; until IsSquare(b + 1);
  y := SquareRoot(b + 1);
  E2 := EllipticCurve([Fpk!0, b]);
  G2 := E2![1, y];
until IsZero(q*G2);

E1 := BaseChange(E1,Fpk);
E2;

g, phiisomorphism := IsIsomorphic(E2, E1);
g;
phiisomorphism;
phi := IsomorphismToIsogeny(phiisomorphism);
"phi:", phi;

N := Resultant(s^k - 1, s^2 - t*s + p);  // number of points of
   E2 over big field
Cofac := N div q^2;
Q := Cofac*Random(E2(Fpk));
while IsZero(q*Q) eq false do
Q := Cofac*Random(E2(Fpk));
end while;
P := E1!P;
"Alternative pairing of P and Q:", WeilPairing(P,phi(Q),q);
```

Appendix J

Extension Issuing protocol

The Magma-code to implement the extension of the Issuing protocol. (Some results of the implementation of the alternative pairing are shortened, because the result of some computations will not change.)

```
Zs<s> := PolynomialRing(Integers());
p := 249785771109578071340305660639915127509902072 4723;
q := 249785771109578071340305502593791761863447 3494829;
"p=",p;
"q=",q;
Fp := FiniteField(p);
y := SquareRoot(Fp!(19));
E1 := EllipticCurve([Fp!0, 18]);
G1 := E1![1, y];
"E1=",E1;
t := p + 1 - q;
k := 12; // security multiplier
Fpk := GF(p, k);
P := E1(Fp)!G1;
E2 := EllipticCurve([Fpk!0, 28]);
E1 := BaseChange(E1,Fpk);
"E2=",E2;
N := Resultant(s^k - 1, s^2 - t*s + p);  // number of points of E2 over
   big field
Cofac := N div q^2;
Q := Cofac*Random(E2(Fpk));
while IsZero(q*Q) eq false or Q in PointsAtInfinity(E2) do
```

```
Q := Cofac*Random(E2(Fpk));
end while;

// Start extension issuing protocol
Fq:=FiniteField(q);
m := 10;
x := Matrix( 1, m, [1..m by 1]);
for i in [1..m] do
x[1,i] := Random(q-1);
while x[1,i] eq 0 do
x[1,i] := Random(q-1);
end while;
end for;
y := Random(q-1);
while y eq 0 do
y := Random(q-1);
end while;
z := Random(q-1);
while z eq 0 do
z := Random(q-1);
end while;

Y := y*P;
Z := z*P;
X := [ P : i in [1..m] ];
for i in [1..m] do
X[i] := x[1,i]*P;
end for;

//Choosing attribute values
av := Matrix( 1, m, [1..m by 1]);
for i in [1..m] do
av[1,i] := i;
end for;
av0 := Random(q-1);

x0 := Random(q-1);
while x0 eq 0 do
x0 := Random(q-1);
```

```
end while;
X0 := x0*P;
C := av0*X0+&+[ av[1,i]*X[i] : i in [1..m] ];

//Creating Paillier setup
p1 := RandomPrime(95);
p2 := q;
n := p1*p2;
lambda := Lcm(p1-1,p2-1);
Integers := Integers();
nZ2 := ideal<Integers | n^2>;
Zn2 := quo<Integers|nZ2>;
g := Random(Zn2);
gpowerlambdaasinteger := Integers!(g^lambda);
while Gcd(g,n^2) ne 1 or Gcd(Order(g),n) ne n or
   Gcd((gpowerlambdaasinteger -1) div n,n) ne 1 do
   g := Random(Zn2);
   gpowerlambdaasinteger := Integers!(g^lambda);
end while;
rho := Random(n-1);
while Gcd(rho,n) ne 1 do
   rho := Random(n-1);
end while;
rhotilde := Zn2!rho;

beta := Random(n-1);
r := Random(n-1);
bbar :=(g^beta)*(rhotilde^n);
rbar :=(g^(beta*r))*(rhotilde^n);
av0bar :=(g^((beta*av0) mod n))*(rhotilde^n);

gamma := Random(q-1);
while Gcd(gamma,q) ne 1 do
   gamma := Random(q-1);
end while;
ka := y+&+[ av[1,i]*x[1,i] : i in [1..m] ];
nZ := ideal<Integers | n>;
Zn := quo<Integers|nZ>;
z := Zn!z;
```

```
z := Integers!z;
x0 := Zn!x0;
x0 := Integers!x0;
ka := Zn!ka;
ka := Integers!ka;
gamma := Zn!gamma;
gamma := Integers!gamma;
d := rbar^z*av0bar^x0*bbar^ka*(g^gamma*(rhotilde^n));

dpowerlambdaasinteger := Integers!(d^lambda);
inversedenominator := Modinv((gpowerlambdaasinteger -1) div n,n);
sdecbar:= (((dpowerlambdaasinteger - 1) div n)*inversedenominator) mod n;
sdec := sdecbar-(gamma mod q) mod q;
Stilde := (Modinv(sdec,n) mod q)*Q;

S := beta*Stilde;
g, phiisomorphism := IsIsomorphic(E2, E1);
phi := IsomorphismToIsogeny(phiisomorphism);

P := E1!P;
WeilpPphiQ := WeilPairing(P,phi(Q),q);
WeilpYCrZphiS := WeilPairing(E1!(Y+C+r*Z),phi(S),q);
"Alternative pairing of P and Q:", WeilpPphiQ;
"Alternative pairing of Y+C+rZ and S:", WeilpYCrZphiS;
if WeilpPphiQ eq WeilpYCrZphiS then
   "Valid signature";
else
   "Invalid signature";
end if;
```

Appendix K

Extension Showing protocol

The Magma-code to implement the extension of the Showing protocol. (Some results of the implementation of the alternative pairing and the extended Issiung protocol are shortened, because the result of some computations will not change.)

```
Zs<s> := PolynomialRing(Integers());
p := 249785771109578071340305660639915127509902072 4723;
q := 249785771109578071340305502593791761863447 3494829;
"p=",p;
"q=",q;
Fp := FiniteField(p);
y := SquareRoot(Fp!(19));
E1 := EllipticCurve([Fp!0, 18]);
G1 := E1![1, y];
"E1=",E1;
t := p + 1 - q;
k := 12; // security multiplier
Fpk := GF(p, k);
P := E1(Fp)!G1;
E2 := EllipticCurve([Fpk!0, 28]);
E1 := BaseChange(E1,Fpk);
"E2=",E2;
N := Resultant(s^k - 1, s^2 - t*s + p); // number of points of E2 over
   big field
Cofac := N div q^2;
Q := Cofac*Random(E2(Fpk));
while IsZero(q*Q) eq false or Q in PointsAtInfinity(E2) do
```

```
Q := Cofac*Random(E2(Fpk));
end while;

// Start extension issuing protocol
Fq:=FiniteField(q);
m := 10;
x := Matrix( 1, m, [1..m by 1]);
for i in [1..m] do
x[1,i] := Random(q-1);
while x[1,i] eq 0 do
x[1,i] := Random(q-1);
end while;
end for;
y := Random(q-1);
while y eq 0 do
y := Random(q-1);
end while;
z := Random(q-1);
while z eq 0 do
z := Random(q-1);
end while;

Y := y*P;
Z := z*P;
X := [ P : i in [1..m] ];
for i in [1..m] do
X[i] := x[1,i]*P;
end for;

//Choosing attribute values
av := Matrix( 1, m, [1..m by 1]);
for i in [1..m] do
av[1,i] := i;
end for;
av0 := Random(q-1);

x0 := Random(q-1);
while x0 eq 0 do
x0 := Random(q-1);
```

```
end while;
X0 := x0*P;
C := av0*X0+&+[ av[1,i]*X[i] : i in [1..m] ];

//Creating Paillier setup
p1 := RandomPrime(90);
p2 := q;
n := p1*p2;
lambda := Lcm(p1-1,p2-1);
Integers := Integers();
nZ2 := ideal<Integers | n^2>;
Zn2 := quo<Integers|nZ2>;
g := Random(Zn2);
gpowerlambdaasinteger := Integers!(g^lambda);
while Gcd(g,n^2) ne 1 or Gcd(Order(g),n) ne n or
   Gcd((gpowerlambdaasinteger -1) div n,n) ne 1 do
g := Random(Zn2);
gpowerlambdaasinteger := Integers!(g^lambda);
end while;
rho := Random(n-1);
while Gcd(rho,n) ne 1 do
rho := Random(n-1);
end while;
rhotilde := Zn2!rho;

beta := Random(n-1);
r := Random(n-1);
bbar :=(g^beta)*(rhotilde^n);
rbar :=(g^(beta*r))*(rhotilde^n);
av0bar :=(g^((beta*av0) mod n))*(rhotilde^n);

gamma := Random(q-1);
while Gcd(gamma,q) ne 1 do
gamma := Random(q-1);
end while;
ka := y+&+[ av[1,i]*x[1,i] : i in [1..m] ];
nZ := ideal<Integers | n>;
Zn := quo<Integers|nZ>;
z := Zn!z;
```

```
z := Integers!z;
x0 := Zn!x0;
x0 := Integers!x0;
ka := Zn!ka;
ka := Integers!ka;
gamma := Zn!gamma;
gamma := Integers!gamma;
d := rbar^z*av0bar^x0*bbar^ka*(g^gamma*(rhotilde^n));

dpowerlambdaasinteger := Integers!(d^lambda);
inversedenominator := Modinv((gpowerlambdaasinteger -1) div n,n);
sdecbar:= (((dpowerlambdaasinteger - 1) div n)*inversedenominator) mod n;
sdec := sdecbar-(gamma mod q) mod q;
Stilde := (Modinv(sdec,n) mod q)*Q;

S := beta*Stilde;
g, phiisomorphism := IsIsomorphic(E2, E1);
phi := IsomorphismToIsogeny(phiisomorphism);

P := E1!P;

//Start of extended showing protocol
D := [ i : i in [1..m by 2] ];
CC := Matrix( 1, m-#D, [1..(m-#D) by 1]);
a := 1;
for i in [1..m] do
v := i notin D;
if v eq true then
CC[1,a] := i;
a := a+1;
end if;
end for;

h := [1..(m-#D) by 1];
for i in [1..(m-#D)] do
h[i] := CC[1,i];
end for;
CC := h;
```

```
nu := Random(q-1);
while nu eq 0 do
   nu := Random(q-1);
end while;

Ctilde := C+nu*Z;
CD := &+[ av[1,i]*X[i] : i in D ];

sigma := Random(q-1);
while sigma eq 0 do
   sigma := Random(q-1);
end while;

Stilde := sigma*S;
rtilde := r-nu;
R := rtilde*S;
Rtilde := rtilde*Stilde;
Qtilde := sigma*Q;

if WeilPairing(E1!(Y+Ctilde),phi(Stilde),q)*
   WeilPairing(E1!Z,phi(Rtilde),q) eq WeilPairing(P,phi(Qtilde),q) then
   "Signature accepted";
else
   "Signature not accepted";
end if;

Q in PointsAtInfinity(E2); //This should be 'false'
av0accent := Random(q-1);
nuaccent := Random(q-1);
avaccent := Matrix( 1, m, [1..m by 1]);
for i in [1..m] do
if i in CC then
avaccent[1,i] := i;
end if;
end for;

N := nuaccent*Z+av0accent*X0+&+[avaccent[1,i]*X[i] : i in CC ];
gamma2 := Random(q-1);
while gamma2 eq 0 do
```

```
   gamma2 := Random(q-1);
end while;

rnu := gamma2*nu+nuaccent;
rav0 := gamma2*av0+av0accent;
rav := Matrix( 1, m, [1..m by 1]);
for i in CC do
   rav[1,i] := gamma2*av[1,i]+avaccent[1,i];
end for;

if N eq rnu*Z+rav0*X0+&+[rav[1,i]*X[i] : i in CC ]-gamma2*(Ctilde-CD)
   then
   "Proof of knowledge over all the attribute values";
else
   "No proof of knowledge over all the attribute values";
end if;
```

Bibliography

Balasubramanian, R., & Koblitz, N. (1998). The Improbability That an Elliptic Curve Has Subexponential Discrete Log Problem under the MenezesOkamotoVanstone Algorithm. *Journal of Cryptology*, 11:141145.

Boneh, D., & Boyen, X. (2008). Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21:149177.

Brands, S. A. (2000). *Rethinking public key infrastructures and digital certificates: building in privacy.* Cambridge, MA: MIT Press.

Costello, C. (2012). Fast Formulas for Computing Cryptographic Pairings. (Unpublished thesis). University of Queensland.

De Feo, L., Jao, D., & Plût, J. (2012). *Isogeny graphs in cryptography.* (Presentation). Universit*é* de Versailles and University of Waterloo.

ElGamal T. (1985). A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4), 469-472.

EPFL-*É*cole polytechnique f*é*d*é*rale de Lausanne. Laboratory for cryptologic algorithms - LACAL , http://lacal.epfl.ch/112bit_prime (consulted on 22 December 2013).

Fontein, F. (2005). Elliptic Curves over Rings with a Point of View on Cryptography and Factoring. (Unpublished thesis). Universit*ä*t Oldenburg.

Galbraith, S. D. (2012). *Mathematics of public key cryptography.* Cambridge University Press.

Galbraith, S. D, Paterson, K. G., & Smart, N. P. (2008). Pairings for Cryptographers. *Discrete Applied Mathematics*, 156(16), 3113-3121.

Hoepman, J., & Lueks, W. (2012). Towards efficiently revocable self-blindable credentials. (Unpublished manuscript).

Joux, A. (2002). Algorithmic Number Theory (5th International Symposium, ANTS-V) Lecture Notes in Computer Science Volume 2369. *The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems* (pp. 20-32). London, Springer Berlin Heidelberg.

Joux, A. (2013). https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind1305&L=NMBRTHRY&F=&S=&P=3034 (consulted on 12 December 2013).

Lenstra, H. W. (1987). Factoring Integers with Elliptic Curves. *The Annals of Mathematics*, Second Series, 126(3), 649-673.

Magma, http://magma.maths.usyd.edu.au/calc/.

Paillier, P. (1999). In Stern, J. (Ed) Advances in Cryptology - EUROCRYPT '99, volume 1592 of Lecture Notes in Computer Science. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes* (pp. 223-238). Springer-Verlag.

Pointcheval, D., & Stern, J (1996). In Kim, K., & Matsumoto, T. (Eds.) Advances in Cryptology - ASIACRYPT '96, volume 1163 of Lecture Notes in Computer Science. *Provably secure blind signature schemes* (pp. 252-265). Springer-Verlag.

Schnorr, C. D. (1990). In Brassard, G. (Ed.) Advances in Cryptology - CRYPTO '89, volume 435 of Lecture Notes in Computer Science. *Efficient Identification and Signatures for Smart Cards* (pp. 239-252). Springer-Verlag Berlin Heidelberg.

Silverman, J. H. (2009). *The Arithmetic of Elliptic Curves* (2nd ed.). Springer-Verlag.

Sutherland, A. V. (2012). *Isogeny volcanoes: a computational perspective.* (Presentation). Massachusetts Institute of Technology.

Tate, J (1966). Endomorphisms of Abelian Varieties over Finite Fields. *Inventiones mathematicae*, volume 2,134-144.

Tezcan, C. (2009). Random Self-reducibility of the Discrete Logarithm Problem for Genus 2 Curves. (Candidacy exam write-up, Ecole Polytechnique Federale de Lausanne, Faculté I&C, Laboratory for cryptologic algorithms).

Weitenberg, E. (2012). Providing unlinkability of transactions with a single token in U-Prove. (Unpublished thesis). Universiteit van Groningen, in coorporation with TNO.