Visualisation of Modern Key Exchange Schemes for more than two Parties in CrypTool and their Security Analysis

Cristina Onete

Technische Universiteit Eindhoven

October 7, 2008

Author: Cristina Onete
E-mail: cristina.onete@gmail.com
Supervisors:

- Prof. Dr. Tanja Lange, Technische Universiteit Eindhoven
- Prof. Bernhard Esslinger, Universität Siegen

Institution: Technische Universiteit Eindhoven
Faculty: Wiskunde en Informatica
Study: Industrial & Applied Mathematics
Specialisation: Discrete & Applied Mathematics

Abstract

Key exchange is a prerequisite of most types of secure communication. Its purpose is to allow an arbitrarily large number of users n to communicate securely over an insecure line (the Internet). The Diffie-Hellman key exchange protocol is the most famous protocol that achieves key exchange for the case of two users only; this protocol was originally implemented only in finite fields of prime order.

It is the purpose of this thesis to describe several other settings in which cryptography may be implemented: namely elliptic curves and pairings on elliptic curves. The translation of the Diffie-Hellman protocol to elliptic curves is explained and the two settings are compared. For the pairing setting, the Tate and Weil pairings are presented, and a basic overview of pairing-friendly curves with embedding degrees $k = 2$ and $k = 12$ is given. As a basic arrangement for three-partite key exchange in a pairing setting, a protocol by Joux is described and analysed. Finally, for arbitrary values of n, two versions of the BD I (named after its inventors Burmester and Desmedt) key exchange protocol and the BD II key exchange protocol are presented in finite fields and in the pairing setting. New additions to the protocols already described in the literature are: a slight extension for the BD II protocol in the pairing setting to include cases in which users might have only one child instead of two on one of its branches, and a turn-based BD I protocol for the pairing setting. For each of the protocols in the pairing setting, an explicit method has been given of how to modify the protocol for elliptic curves that do not support distortion maps.

It was also the intention of this master project to implement multi-partite key exchange in JCrypTool – a didactic tool that demonstrates various cryptographic and security-related topics. Some details of this implementation are also presented in this thesis.

Contents

Chapter 1

Introduction

Key exchange is an important step in any type of secure communication over an insecure line. Suppose that we have a situation where several users want to securely communicate with each other. An example of secure internet communication which occurs increasingly often is internet-shopping. In such cases, a user – the home-shopper – attempts to establish a secure connection to a server. The server must prove its identity to the home-shopper, while the remote user must be provided sufficient security to send sensitive bank information across the internet.

Another example of secure communication may occur when various parties want to hold an online conference. In this case, all the parties need to be able to verify the identities of their peers and must have the security that the information they send and receive can only be decrypted by those involved in the conference. The connection between these parties takes place via internet; thus the sent and received information is accessible by eavesdroppers. In this situation, the security of the communication has to be provided by cryptography.

A practical, standardised method of ensuring secure multipartite communication is the TLS protocol as described by [11]. This protocol is currently in use and supported by several applications, such as HTTP, FTP, SMTP, NNTP, or XMPP; in order to work properly, it requires the existence of a reliable information transport protocol, such as TCP. When for example HTTP is used together with TLS, it has as result HTTPS. This protocol is secure against both passive and active adversaries; that is to say, using HTTPS ensures protection against eavesdropping, message tampering, and message forgery. This protocol is used in internet shopping and is identified by the "https" appearing in the URL. A bank transaction via internet banking, or even a payment on www.amazon.com will be made via HTTPS. The protocol needs to be supported by the browser so as to minimise the user's involvement; for the convenience of the communicating parties, it is also necessary for the protocol to be fast. There are three phases in the TLS protocol, the second of which is key exchange:

1. Peer negotiation for algorithm support;
2. Authenticated key exchange;

3. Authenticated symmetric encryption during message exchange.

During authentication, each identity – a server, a user – is bound to a set of private and public keys. An entity must prove its identity by proving it knows the secret key corresponding to its public key. There are different levels of security that can be considered in a TLS protocol; when more parties are required to authenticate themselves, it becomes less likely for an adversary to disrupt communication by impersonation attacks. Ideally, of course, every communicating party needs to authenticate itself. However, each entity that requires authentication needs to have a certificate issued by some trusted authority; for this, each participant is usually involved in a PKI – Public Key Infrastructure (as an alternative to the rather centralised PKI model, one could also use the web of trust model in PGP). HTTPS usually provides communication between only two parties – a user and a server – and the tendency is to minimise user involvement. Therefore, most HTTPS communications will only authenticate the server and not both parties. However, when several users wish to communicate securely, they will all need to be authenticated.

This thesis concerns key exchange in greater detail; an instance of this process occurs as the second phase of the TLS protocol, but it is not restricted to it. Key exchange is a prerequisite of any exchange of information in which symmetric cryptography occurs, and the keys that the users obtain as a consequence are used in symmetric encryption and decryption. The situation we consider is that of more than two parties wishing to communicate securely. If we are to examine TLS, we notice that key exchange is preceded by an initial contact between the underlying structures of the users' web browsers, which ensures peer negotiation for algorithm support and establishes the algorithms that are subsequently used in authentication, key exchange, and cryptography algorithms. For further information on the peer negotiation phase in TLS, we refer to [11].

Key exchange occurs as a second phase of the TLS protocol, and as a first phase in other types of cryptographic communication. This phase is meant to provide each of the parties involved in secure communication with a conference key, which further enables them to use symmetric encryption. Because key exchange is only the first step in secure communication, it must be efficient and secure. Authentication must be provided in order to prevent active attackers, in particular attackers impersonating one of the users, intercepting and sending messages of its own without being detected.

Several key exchange protocols exist and are already in use. For two-party key exchange, the most classical protocol is the Diffie-Hellman key exchange protocol – which will also be presented in the course of this thesis. This is one of the standard key exchange protocols used by TLS. The focus of the present work, however, is the case when more than two participants are considered; we call this multi-partite key exchange. At the moment, several implementations of such protocols exist; the mathematical background for these is presented in chapter 2.

The Burmester Desmedt (BD) protocols provide key exchange for any number n of participants, and can be implemented both in finite fields and in a pairing setting. In finite fields, the BD I protocol arranges the users in a circle; when pairings are considered, the

arrangement is made up of triangular units.

Should the participants to the protocol be organised in a hierarchical structure, such as a binary tree, key exchange could be achieved by means of the BD II protocol. In finite fields, the BD II protocol arranges its users in a binary tree, while in the pairing setting, the users are arranged in a triangle-based tree.

Once all the users know which key exchange protocol suits all of them and is supported by their hardware and software, i.e. their browsers, computers, laptops, or mobile phones, the protocol can be put into use and results in a conference key. This key will then be used for message encryption and authentication.

Bi-partite key exchange is the basis of general key exchange, and the case for two users can be extended to more participants. We show a practical example of how authenticated bi-partite key exchange works, without restricting ourselves to a particular protocol. We can think of this protocol as the one used in TLS. We call the users Alice and Bob, and we assume that they have somehow agreed on protocols $\mathcal{A}_{\text{auth}}$ for authentication, $\mathcal{A}_{\text{kexch}}$ for key exchange, and $\mathcal{A}_{\text{crypt}}$ for further symmetric cryptography.

Phase Two: Key Exchange

Alice		Bob
knows: Sa_{auth}, Pa_{auth} gets: Pb_{auth}		knows: Sb_{auth}, Pb_{auth} gets: Pa_{auth}
generates: Sa_{kexch} computes $\sigma_a = \text{Sign}(Pa_{\text{kexch}}, Sa_{\text{auth}})$ computes: $\text{m1}_{\text{kexch}} = \{Pa_{\text{kexch}}, \sigma_a\}$	$\longrightarrow$	generates: Sb_{kexch} computes $\sigma_b = \text{Sign}(Pb_{\text{kexch}}, Sb_{\text{auth}})$ receives: m1_{kexch} checks: σ_a with Pa_{auth} gets: Pa_{kexch}
receives: m2_{kexch} checks: σ_b with Pb_{auth} gets: Pb_{kexch} computes: K_{kexch}	$\longleftarrow$	computes: $\text{m2}_{\text{kexch}} = \{Pb_{\text{kexch}}, \sigma_b\}$ computes: K_{kexch}

In the key exchange phase, authentication is used with every message sent through public channels. The users are assumed to know the public authentication keys of all the other peers before the beginning of the protocol. This could be done for example by using lists that bind identities with public keys. The key exchange protocol dictates then how the generated secret and public keys are used in finding the conference key K_{exch}. Symmetric cryptography is afterwards used to encrypt and decrypt messages such as t_1.

As stated before, it is the purpose of this thesis to investigate multi-partite key exchange protocols. In particular, the protocols shown in this paper are: Diffie-Hellman, Joux' tripartite key exchange, BD I, and BD II. The algorithms are presented in various settings: finite fields, elliptic curves, and pairings. Moreover, Java implementations of these protocols were provided and incorporated in CrypTool – [8].

CrypTool is an open source project that aims at enhancing worldwide awareness of cryptography and its uses; it is also used as an educational tool in several institutions, such as for example the University College in Utrecht. The Deutsche Bank and several universities are involved in the development of CrypTool; there are also several independent contributors involved. Their combined efforts have led to the development of an open source tool that shows several classic and modern methods of cryptography, amongst which: Caesar ciphers, Enigma, RSA, AES, and various methods of cryptanalysis.

At the moment, two versions of CrypTool are available: one containing implementations in C# and one in Java; the latter is called JCrypTool. It was the aim of this project to integrate key exchange in JCrypTool and provide online help and usable interfaces for the multi-partite key exchange protocols mentioned above. Chapter 7 shows the implementation aspect in more detail. Several mathematical tricks that have been used for the programming are described in the Appendix.

For the readers who are not already familiar with cryptographically relevant notions of group theory, finite fields, elliptic curves, and pairings, we include a brief mathematical background in chapter 2. An overview of several relevant cryptographic settings is shown in chapter 3, and the respective, relevant hard problems that ensure security on these settings are stated in chapter 4. The multi-partite key exchange protocols mentioned above are described and analysed in chapter 5, and details about authentication are further explained in chapter 6. Finally, several conclusions are presented in chapter 8.

Chapter 2

Mathematical Background

In this chapter, a brief overview is given of the more important mathamatical concepts used throughout this paper. The results presented here are also found in the literature, of which we mention in particular [2], [5], and [28].

2.1 Groups

Definition 2.1.1. *A* group *is a set G together with an operation $\circ : G \times G \longrightarrow G$ for which the following properties hold:*

1. ***Closure:*** *For any two elements $a, b \in G$, $a \circ b \in G$;*
2. ***Associativity:*** *For any three elements $a, b, c \in G$, $a \circ (b \circ c) = (a \circ b) \circ c$;*
3. ***Identity element:*** *There exists an element $\mathbf{I} \in G$ such that for any element $a \in G$, $\mathbf{I} \circ a = a \circ \mathbf{I} = a$;*
4. ***Inverse element:*** *For any $a \in G$, there exists $b = \overline{a}$ such that $a \circ b = b \circ a = \mathbf{I}$. The element $\overline{a}$ is called the inverse of a.*

In many cases, a group is denoted by the symbol of the set it is built on. For example, the group of integers modulo p under addition is denoted $\mathbb{F}_p$, rather than $(\mathbb{F}_p, +)$. This short notation will also be used in this thesis, when the context is clear.
Special kinds of groups are Abelian and cyclic groups.

Definition 2.1.2. *A group $(G, \circ)$ is called* Abelian *if additionally for any $a, b \in G$ it holds that $a \circ b = b \circ a$.*

As a matter of notation, we denote the repeated use of the group operation $\circ$ on an element g as g^n, and we call it exponentiation. Therefore, $g^3 = g \circ g \circ g$, and so forth. We write $\mathbf{I} = g^0$ for any element $g \in G$.

Definition 2.1.3. *A group $(G, \circ)$ is called* cyclic *if there exists an element $g \in G$ such that all elements $h \in G$ can be written as $h = g^k$ for some $k \in \mathbb{Z}$. The element g is then called* generator *of the group.*

We also introduce the notions of element and group order.

Definition 2.1.4. *Let G be a group under the operation $\circ$ and consider an arbitrary element $g \in G$. The* order *of g is the smallest positive integer k such that $g^k = \mathbf{I}$, or infinity.*

Definition 2.1.5. *Let $(G, \circ)$ be a group. The* order of this group *is the cardinality of G, i.e. the number of elements in G.*

A concept that will appear frequently in the subsequent chapters is that of a subgroup.

Definition 2.1.6. *Let $(G, \circ)$ be a group. $H \subseteq G$ is a* subgroup *of G if $(H, \circ)$ is a group.*

We further define the concepts of cosets and quotient groups, as these structures are necessary when defining pairings.

Definition 2.1.7. *Let H be a subgroup of the group $(G, \circ)$. Let $g \in G$ be an arbitrary group element. Then:*

$$\begin{aligned} gH &= \{g \circ h, \quad h \in H\} \text{ and} \\ Hg &= \{h \circ g, \quad h \in H\} \end{aligned}$$

are left *and* right cosets *of H. Clearly, if G is an Abelian group, the two cosets are identical.*

Definition 2.1.8. *A subgroup H of $(G, \circ)$ is* normal *if and only if $gH = Hg$ for all elements $g \in G$.*

Definition 2.1.9. *Let H be a normal subgroup of a group $(G, \circ)$. Then the* quotient group *G/H is the set of all left cosets of H in G; in other words:*

$$G/H = \{gH \ : \ g \in G\}. \tag{2.1}$$

The elements of G/H are sets, and the cardinality of G/H is usually less than the cardinality of G. We define set product as an operation, and mention that it can be used on the quotient group G/H:

Definition 2.1.10. *Let $(G, \circ)$ be a group, and consider $S_1, S_2 \subset G$. We consider the* product *of S_1 and S_2 to be:*

$$S_1 S_2 = \{s_1 \circ s_2 \ , \quad s_1 \in S_1, s_2 \in S_2\}. \tag{2.2}$$

Set product defines a group operation on $G = G/H$. The product of two elements of G/H, $S_1 = g_1H$ and $S_2 = g_2H$, is again an element of G/H as $S_1S_2 = (g_1H)(g_2H) = (g_1 \circ g_2)H$, or $\hat{g}H$ for $\hat{g} := g_1 \circ g_2$.
Some groups can be related to one another by means of special maps called homomorphisms:

Definition 2.1.11. *Let $(G, \circ)$ and $(H, \bullet)$ be two groups. A* group homomorphism *from G to H is a map $\phi : G \longrightarrow H$ such that for all $u, v \in G$, $\phi(u \circ v) = \phi(u) \bullet \phi(v)$.*

Important structures related to homomorphisms are the kernel and the image of the map.

Definition 2.1.12. *Let $\phi : G \longrightarrow H$ be a homomorphism between the two groups $(G, \circ)$ and $(H, \bullet)$. The* kernel *of ϕ is $\mathcal{K}_\phi = \{g \in G : \quad \phi(g) = \mathbf{I}_H\}$; the* image *of the homomorphism is $\mathcal{I}_\phi = \{\phi(g) : \quad g \in G\}$.*

A notion that will be useful in the understanding of finite fields is equivalence. In what follows we define equivalence and several related notions.

Definition 2.1.13. *Let S be a set. An* equivalence relation *is a binary relation $\sim$ that relates two elements of $a, b \in S$ – we say that a is equivalent to b if $a \sim b$. The relation $\sim$ should have the following properties:*

- ***Reflexivity:*** *$a \sim a$;*
- ***Symmetry:*** *$a \sim b$ implies $b \sim a$;*
- ***Transitivity:*** *$a \sim b$ and $b \sim c$ imply $a \sim c$.*

Definition 2.1.14. *Let a be an element in S. An* equivalence class $[a]$ *under the relation $\sim$ is the set of all b such that $a \sim b$. Note that $a \in [b]$ by the reflexivity of the relation $\sim$.*

Definition 2.1.15. *The set of all possible equivalence classes $[s]$ of S under $\sim$ is the quotient set of S by $\sim$.*

2.2 Fields and Finite Fields

Definition 2.2.1. *A* field *is a set $\mathbb{K}$ together with two operations – generally called addition and multiplication and denoted $+$ and $\cdot$ – such that the following properties hold:*

- ***Closure under $+$ and $\cdot$:*** *for all $a, b \in \mathbb{K}$, $a + b \in K$ and $a \cdot b \in \mathbb{K}$;*
- ***Associativity under $+$ and $\cdot$:*** *for all $a, b, c \in \mathbb{K}$, $(a + b) + c = a + (b + c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$;*

- ***Commutativity of*** $+$ ***and*** $\cdot$*: for all* $a, b \in \mathbb{K}$*, it holds that* $a + b = b + a$ *and* $a \cdot b = b \cdot a$*;*
- ***Identity elements for*** $+$ ***and*** $\cdot$*: there exist elements* $\mathbf{0} \neq \mathbf{I}$ *such that for all* $a \in \mathbb{K}$*:* $a + \mathbf{0} = \mathbf{0} + a = a$ *and* $a \cdot \mathbf{I} = \mathbf{I} \cdot a = a$*;*
- ***Inverses under*** $+$*: all elements* $a \in \mathbb{K}$ *have an inverse* $(-a)$ *under addition;*
- ***Inverses under*** $\cdot$*: all elements* $a \neq \mathbf{0}$ *have an inverse* a^{-1} *under multiplication;*
- ***Distributivity:*** *multiplication is distributive with respect to addition. For all elements* $a, b, c \in \mathbb{K}$*, it holds that* $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$*.*

In other words, $\mathbb{K}$ is an Abelian group under addition, $\mathbb{K} \setminus \{\mathbf{0}\}$ is an Abelian group under multiplication, and the law of distributivity holds.
A particular type of field is a finite field.

Definition 2.2.2. *A* finite field *is a field that has finitely many elements.*

In order to correlate structures, such as groups and fields, we require the concept of an isomorphism. We have already defined group homomorphisms; we now translate this definition to fields as follows:

Definition 2.2.3. *Let* $(\mathbb{K}_1, +, \cdot, \mathbf{0}_{\mathbb{K}_1}, \mathbf{1}_{\mathbb{K}_1})$ *and* $(\mathbb{K}_2, \oplus, \odot, \mathbf{0}_{\mathbb{K}_2}, \mathbf{1}_{\mathbb{K}_2})$ *be two fields. A* field homomorphism *is a map* $f : \mathbb{K}_1 \longrightarrow \mathbb{K}_2$ *with the following properties:*

- *For* $a, b \in \mathbb{K}_1$*,* $f(a + b) = f(a) \oplus f(b)$*;*
- *For* $a, b \in \mathbb{K}_2$*,* $f(a \cdot b) = f(a) \odot f(b)$*;*
- $f(\mathbf{0}_{\mathbb{K}_1}) = \mathbf{0}_{\mathbb{K}_2}$ *and* $f(\mathbf{1}_{\mathbb{K}_1}) = \mathbf{1}_{\mathbb{K}_2}$*.*

Definition 2.2.4. *Let* $\mathbb{K}_1$ *and* $\mathbb{K}_2$ *be two fields as above. An* isomorphism ϕ *is a bijective map such that both* ϕ *and* ϕ^{-1} *are field homomorphisms.*

The order of a finite field is either a prime p, or a prime power p^k for some positive integer k. The prime p is called the characteristic of the finite field. For every prime power p^k, there exists exactly one finite field of order p^k (up to isomorphism).
Consider a finite field with p elements, where p is a prime number. This finite field is isomorphic to $\mathbb{Z}/p\mathbb{Z}$ and we use $\mathbb{F}_p$ to denote it. The isomorphism provides a representation of $\mathbb{F}_p$ as equivalence classes, where $a \sim b \Leftrightarrow a \equiv b \bmod p \Leftrightarrow \exists m \in \mathbb{Z}$ s.t. $a - b = mp$. For ease of notation, we generally represent elements of $\mathbb{F}_p$ by the smallest positive element in the equivalence class.

Definition 2.2.5. *Let* $(\mathbb{L}, +, \cdot, \mathbf{0}, \mathbf{1})$ *be a field. If there exists a subset* $\mathbb{K} \subset \mathbb{L}$ *such that* $(\mathbb{K}, +, \cdot, \mathbf{0}, \mathbf{1})$ *is a field, then we call* $\mathbb{L}$ *an* extension field *of* $\mathbb{K}$ *and we write* $\mathbb{L}/\mathbb{K}$*.*

The finite field with p^k elements is an extension field of $\mathbb{F}_p$. They can be constructed from the ring of polynomials over $\mathbb{F}_p$, which is denoted $\mathbb{F}_p[X]$, by considering an irreducible polynomial $f \in \mathbb{F}_p[X]$ of degree k. The elements of $\mathbb{F}_{p^k}$ are equivalence classes in $\mathbb{F}_p[X]$ defined on the relation $\sim$ such that $a \sim b$ if and only if $a - b = gf$, where $g \in \mathbb{F}_p[X]$. Each of these classes is represented by a single element, which we call a representative. We will represent the elements of $\mathbb{F}_{p^k}$ by the smallest degree element in the equivalence class.
A particular kind of extension is an algebraic extension.

Definition 2.2.6. *Let $\mathbb{L}$ be an extension field of $\mathbb{K}$. The extension $\mathbb{L}/\mathbb{K}$ is* algebraic *if every element of $\mathbb{L}$ is algebraic over $\mathbb{K}$, that is to say every element of the field $\mathbb{L}$ is a root of a non-zero polynomial f with coefficients in $\mathbb{K}$.*

Definition 2.2.7. *A field $\mathbb{K}$ is* algebraically closed *if every polynomial $a \in \mathbb{K}[X]$ with $\deg(a) \geq 1$ has a root in $\mathbb{K}$.*

Definition 2.2.8. *Let $\mathbb{K}$ be a field. The* algebraic closure $\overline{\mathbb{K}}$ *is the smallest algebraic extension of $\mathbb{K}$ that is algebraically closed.*

2.3 Elliptic Curves

Definition 2.3.1. *Let $\mathbb{K}$ be a field. An* elliptic curve over this field *is a curve defined by the following affine equation:*

$$E : \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6. \tag{2.3}$$

Here, $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$ are taken such that for any point $P = (x_1, y_1) \in E$ with $x_1, y_1 \in \overline{\mathbb{K}}$ the partial derivatives $2y_1 + a_1x_1 + a_2$ and $3x_1^2 + 2a_2x_1 + a_4 - a_1y_1$ do not vanish simultaneously. This condition ensures that the curve is nonsingular, or in other words it is smooth. We identify the elliptic curve with the points on it over $\overline{\mathbb{K}}$:

$$E := \{(x, y) \in \overline{\mathbb{K}} \times \overline{\mathbb{K}} \text{ s.t. } \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\mathcal{O}\} \ . \tag{2.4}$$

In this equation, $\mathcal{O}$ is the point at infinity.

Definition 2.3.2. *Let E be an elliptic curve defined over the field K with equation as in (2.3). Its discriminant is defined as:*

$$\Delta = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6. \tag{2.5}$$

Herein, we have: $b_2 = a_1^2 + 4a_2$, $b_4 = 2a_4 + a_1a_3$, $b_6 = a_3^2 + 4a_6$, *and* $b_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2$.

An elliptic curve is nonsingular if $\Delta \neq 0$. We consider the $\mathbb{K}$-rational points on the curve E.

Definition 2.3.3. *Let E be an elliptic curve defined over some field $\mathbb{K}$. Let $\mathbb{L} \supseteq \mathbb{K}$ be an algebraic extension of $\mathbb{K}$. The set of $\mathbb{L}$-rational points of this curve is:*

$$E(\mathbb{L}) = \{(x, y) \in E : \quad x, y \in \mathbb{L}\} \cup \{\mathcal{O}\} \, . \tag{2.6}$$

We denote the size of the set of $\mathbb{L}$-rational points on the elliptic curve by $\#E(\mathbb{L})$.

Theorem 2.3.1. *Let E be an elliptic curve defined over the finite field $\mathbb{K} = \mathbb{F}_q$. Then the number $\#E(\mathbb{K})$ of $\mathbb{K}$-rational points are bounded from above and below as follows:*

$$|\#E(\mathbb{K}) - (q + 1)| \leq 2g\sqrt{q} \, . \tag{2.7}$$

We can define a group operation on the set of points on the elliptic curve E. We write the group additively, and thus speak of scalar multiplication instead of exponentiation. We can define addition in two ways: by means of the chord-and-tangent method or by means of divisors. In what follows, we explain chord-and-tangent point addition, while some notions on divisors are given later.
Consider points $P, Q \in E$. By means of, for instance, Bézout's theorem, we can see that, if we take the point at infinity into consideration as well, a line (a curve of degree 1) will intersect an elliptic curve (of degree 3) in exactly $1 * 3 = 3$ points. Thus, if we draw a straight line between two points, the line will intersect the elliptic curve in another point (which could be the point at infinity as well).

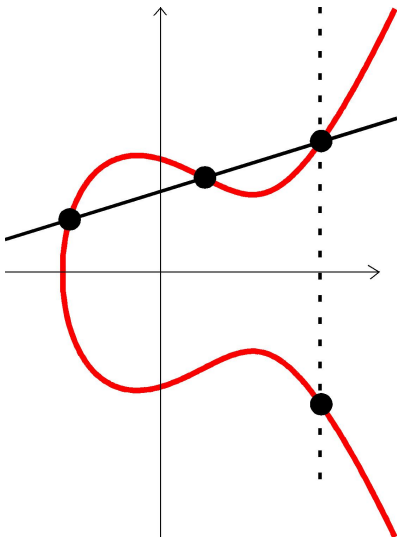

Figure 2.1: A straight line intersects an elliptic curve in 3 points

If the characteristic of the field is greater than 3, an elliptic curve is isomorphic to a curve in short Weierstrass form: $y^2 = x^3 + ax + b$. For such curves therefore, if the point (x, y) is on the curve, then the point $(x, -y)$ is also on the curve and the graph of the curve is symmetric with respect to the x-axis. Figure 2.1 shows such an elliptic curve over the real numbers $\mathbb{R}$. The addition $P + Q$ is defined as follows: we draw the straight line between P and Q, and find the point S where the straight line intersects the curve for the third time. Let the coordinates of point S be (x_S, y_S). Then $P + Q = R$ for $R = (x_S, -y_S)$.

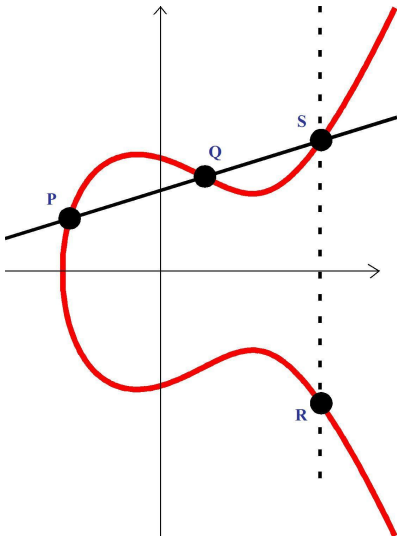

Figure 2.2: Point Addition

If $P = Q$, $P + Q = P + P$ is called point doubling, and it works like addition with the intersecting line through P and Q replaced by the tangent to E at P. If the line is vertical in any of the cases, the result is the point at infinity. If $Q = \mathcal{O}$, adding a point P to Q involves drawing the vertical line through P (this is the line between P and $\mathcal{O}$); the third point of intersection is therefore the point with coordinates $(x_P, -y_P)$, and reflecting this point through the x-axis will give P. Thus: $P + \mathcal{O} = P$. If $P = Q = \mathcal{O}$, then $P + Q = \mathcal{O}$. Consider the example of an elliptic curve with equation $y^2 = x^3 + ax + b$ and $P = (x_P, y_P)$, $Q = (x_Q, y_Q) \in E$ with $P \neq \pm Q$, the slope of the line through P and Q is given by: $\lambda = \frac{y_Q - y_P}{x_Q - x_P}$. The straight line with slope λ going through the point P has equation: $y - y_P = \lambda(x - x_P)$, which is equivalent to $y = \lambda(x - x_P) + y_P$. The intersection between this line and the elliptic curve will give three possible solutions for x: x_P, x_Q, and $x_S = \lambda^2 - x_P - x_Q$. This can be shown as follows: equate $y^2 = (\lambda(x - x_P) + y_P)^2 = x^3 + ax + b$. Then $x^3 - (\lambda(x - x_P) + y_P)^2 + ax + b = 0$, which is a third degree equation and thus has three solutions: x_P, x_Q, and x_S. We write: $x^3 - (\lambda(x - x_P) + y_P)^2 + ax + b = (x - x_P)(x - x_Q)(x - x_S)$ and equate the coefficient of x^2 on both sides of the equation: $-\lambda^2 = -(x_P + x_Q + x_S)$. Thus $x_S = \lambda^2 - x_P - x_Q$. Clearly now, the y-coordinate of the point of intersection of the line and the elliptic curve is $y_S = \lambda(x_R - x_P) + y_P$. The chord-and-tangent rule now indicates that the point $R = P + Q$ has coordinates $x_R = x_S$ and $y_R = -y_S$.

We can prove that the structure $(E, +, \mathcal{O})$ with $+$ denoting the addition defined previously and $\mathcal{O}$ the point at infinity is a group. Indeed, it holds that:

- **Closure**: if $P, Q \in E$, then $P + Q \in E$;
- **Associativity**: for $P, Q, R \in E$, it holds that $(P + Q) + R = P + (Q + R)$;
- **Identity element**: the point at infinity $\mathcal{O}$ is the neutral element, since for all $P \in E$, we have: $P + \mathcal{O} = \mathcal{O} + P = P$;
- **Inverse element**: $\forall\ P = (x_P, y_P) \in E\ \ \exists\ (-P) \in E$ s.t. $P + (-P) = \mathcal{O}$, with $\mathcal{O}$ being the point at infinity. We have: $(-P) = (x_P, -y_P)$ in general, and $-\mathcal{O} = \mathcal{O}$.

As stated before, point doubling represents the addition of a point to itself; it is done by drawing a tangent to the curve at point P and then finding the third point of intersection.
The slope of the tangent to the curve is found by implicit differentiation of the short Weierstrass equation, giving:

$$\lambda = \frac{\mathrm{d}y}{\mathrm{d}x} = \frac{3x^2 + a}{2y}. \tag{2.8}$$

The equation of the line with slope λ going through the point P is, as given above, $y = \lambda(x - x_P) + y_P$. Just as before, we find the x coordinate of $R = 2P$ to be: $x_R = \lambda^2 - 2x_P$ and $y_R = \lambda(x_P - x_R) - y_P$.
It is possible that either the tangent to the curve at point P or the line between points P and Q is vertical. In both case, the third point of intersection is $\mathcal{O}$ and the result of the doubling and respectively of the addition is $\mathcal{O}$ itself – the point at infinity.
Depending on the application for which elliptic curves are needed, one might choose to work with *Jacobian coordinates* instead of affine coordinates – see for example [17] or [1]. Jacobian coordinates are useful when the cost of a field inversion is much higher than the cost of a field multiplication, as working with Jacobian coordinates will exchange inversions for multiplications. A point with Jacobian coordinates (X, Y, Z) represents the affine point (x, y) with $x = \frac{X}{Z^2}$ and $y = \frac{Y}{Z^3}$. Efficient algorithms for point addition and doubling in Jacobian coordinates may be found in [14]. Apart from the regular Jacobian addition and point doubling, one can also use mixed addition, that is the addition of a point in Jacobian coordinates with a point with affine coordinates, both points different from the point at infinity. A possibility for mixed addition is to follow the algorithm for regular Jacobian addition, with the single exception that the Z-coordinate of the affine point is taken by default as 1.
We have already defined the notion of group order in the previous sections. Having proved that the points on an elliptic curve form a group under point addition, we can define the order of a point $P \in E$.

Definition 2.3.4. *Let E be an elliptic curve defined over a field $\mathbb{K}$, and consider a $\mathbb{K}$-rational point $P \in E(\mathbb{K})$. The order of P is the smallest integer k such that $kP = \mathcal{O}$.*

We connect the notion of point order to that of torsion points.

Definition 2.3.5. *Let $E(\mathbb{K})$ be an elliptic curve on a field $\mathbb{K}$ and l an integer. An* l-torsion point *is a point for which $lP = \mathcal{O}$. Therefore, the order of P must be a divisor of the integer l.*

For applications in cryptography, one uses a prime integer l. We denote the set of all $\mathbb{K}$-rational l-torsion points on the curve by $E(\mathbb{K})[l]$. We have proved that the points on the elliptic curve form a group under addition; we now consider a subgroup of the $\mathbb{K}$-rational points on the elliptic curve E.

Lemma 2.3.1. *Let $\mathbb{K} = \mathbb{F}_p$, the finite field of integers modulo a prime p. Let l be a prime number. Then the set $E(\mathbb{F}_p)[l]$ forms a group under the operation of point addition.*

Proof: We point out that if l does not divide the number of points on the elliptic curve, the set $E(\mathbb{F}_p)[l] = \{\mathcal{O}\}$, which is the trivial group under addition. This follows because any l-torsion point on E other than $\mathcal{O}$ has order $l \neq 1$, which must divide the number of $\mathbb{K}$-rational points on the elliptic curve.

As we have proven the points on an elliptic curve to form a group under the operation of point addition, it holds that also l-torsion points in particular are associative under point addition. Similarly, we have that $\mathcal{O}$ is the identity element of point addition and that each point $P \in E$ has an inverse $(-P) \in E$. It remains to prove the closure of l-torsion points under point addition, and the fact that both $\mathcal{O}$ and the inverse of each $P \in E(\mathbb{F}_p)[l]$ are themselves l-torsion points. We use several properties of point addition and prove this:

1. **Closure:** Let $P, Q \in E(\mathbb{F}_p)[l]$ be l-torsion points on the elliptic curve $E(\mathbb{F}_p)$. It holds that $lP = \mathcal{O}$ and $lQ = \mathcal{O}$. Then: $\mathcal{O} = lP + lQ = l(P + Q)$ and $P + Q \in E(\mathbb{F}_p)[l]$.

2. **Identity Element:** $\mathcal{O} \in E(\mathbb{F}_p)[l]$ as $l\mathcal{O} = \mathcal{O} \in E(\mathbb{F}_p)[l]$.

3. **Inverse:** Let $P \in E(\mathbb{F}_p)[l]$ be a point on the elliptic curve and let $(-P)$ be its inverse under addition. It holds that $l(-P) = -lP = -\mathcal{O} = \mathcal{O}$. Therefore, the inverse of an l-torsion point is itself an l-torsion point.

As preparation for the notion of pairings, which will be defined in the following section, we give a short overview of divisors and recommend [1] to the interested reader. In what follows, we consider an elliptic curve E over a field $\mathbb{K}$ and denote the set of $\mathbb{K}$-rational points on E by $E(\mathbb{K})$.

Definition 2.3.6. *A* divisor *is a finite formal sum of points on the elliptic curve. $\mathcal{D} = \sum_{P \in E} a_P(P)$ with $a_P \in \mathbb{Z}$ and $P \in E$ is a divisor if there are only finitely many a_P's that are non-zero. We write (P) for the divisor consisting of P to differentiate the formal sum of points from the sum introduced in Section 1.1.1. Therefore $P + Q$ gives as a result a point R, while $(P) + (Q)$ is a divisor.*

Divisors on an elliptic curve $E(\mathbb{K})$ form an Abelian group under divisor addition; we will denote this group by Div_E. The addition of two divisors is straight-forward.

Definition 2.3.7. *Let $\mathcal{D}_1 = \sum_{P\in E} a_{P\mathcal{D}_1}(P)$ and $\mathcal{D}_2 = \sum_{P\in E} a_{P\mathcal{D}_2}(P)$ be divisors. Then:*

$$\mathcal{D}_1 \oplus \mathcal{D}_2 = \sum_{P\in E} (a_{P\mathcal{D}_1} + a_{P\mathcal{D}_2})(P). \tag{2.9}$$

The operation $+$ is point addition as defined above; by contrast, divisor addition *is denoted $\oplus$.*

The fact that Div_E is an Abelian group follows because $\mathbb{Z}$ is an Abelian group under addition. We can define a norm for the group of divisors.

Definition 2.3.8. *The* degree function $\deg : \mathrm{Div}_E \longrightarrow \mathbb{Z}$ *is defined as:* $\deg(\mathcal{D}) = \sum_{P\in E} a_P$ *for* $\mathcal{D} = \sum_{P\in E} a_P(P)$. *The map* deg *is a homomorphism of groups and its kernel must be a group itself – the group of divisors that have degree* 0. *We denote this group by* Div_E^0.

We further define functions on elliptic curves.

Definition 2.3.9. *A* function *f on a curve $E(\mathbb{K})$ is a rational function $f(x, y) \in \mathbb{K}(x, y)/E$. We can evaluate a function in a certain point $P = (x_P, y_P) \in E(\mathbb{K})$ and we write: $f(P) = f(x_P, y_P)$.*

Divisors can be assigned to each function. We write $\mathrm{div}(f) = \sum_{P\in E} v_f(P)$, where v_f is the valuation of the function f at P. This valuation gives 0 if $f(P)$ is defined and $f(P) \neq 0$, and it gives the order of the zero at P with positive multiplicity and the order of the pole at P with negative multiplicity otherwise. Let us take as an example a line L that passes through three different points, P, Q, and S. This line will have a zero of order 1 for each of these points, and a pole of order 3 in the point at infinity. Therefore, we have $\mathrm{div}(L) = (P) + (Q) + (S) - 3(\mathcal{O})$.

Definition 2.3.10. *Let $\mathcal{D}$ be a divisor. If there exists a function f such that $\mathcal{D} = \mathrm{div}(f)$, then $\mathcal{D}$ is a* principal divisor.

We notice in the example above that the degree of the divisor is zero. Indeed, this holds for all principal divisors. Without proving it, we state that if we count with multiplicities, there are as many roots as poles in such a function and that these are finitely many. Therefore all the principal divisors are contained in the group Div_E^0.
We furthermore look at multiplication of functions on the elliptic curves, denoted here by $\circ$. The result of function multiplication is of course a new function. It is not difficult to show that for functions f and g it holds that:

$$\mathrm{div}(f \circ g) = \mathrm{div}(f) \oplus \mathrm{div}(g). \tag{2.10}$$

In fact, the set of principal divisors together with addition ($\oplus$) forms a group with neutral element the 0 divisor, i.e. the divisor of a constant function $f \equiv k$, where k is any constant in $\mathbb{K}^*$. We call this group Princ_E. We also know that $\mathrm{Princ}_E \subset \mathrm{Div}_E^0$. We can now define equivalence of divisors.

Definition 2.3.11. *Two divisors $\mathcal{D}_1$ and $\mathcal{D}_2$ are said to be equivalent if $\mathcal{D}_1 - \mathcal{D}_2 \in \mathrm{Princ}_E$. We can thus define equivalence classes in Div_E^0.*

We will now work in the quotient group $\mathrm{Pic}_E^0 = \mathrm{Div}_E^0/\mathrm{Princ}_E$. Point addition can also be related to divisors. We recall the chord-and-tangent point addition:

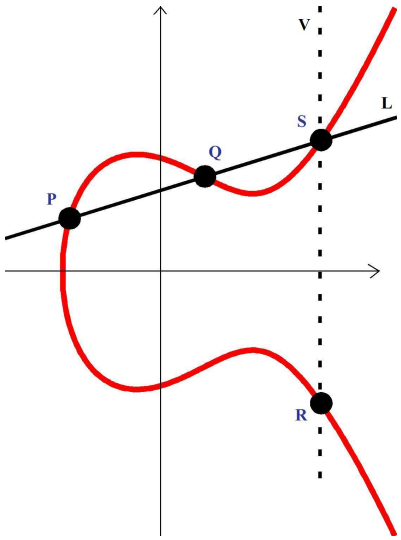

Figure 2.3: Point Addition

The line through P, Q, and S is denoted L, and the vertical line connecting S and R is denoted V. Each of these lines represents a function. It holds that:

$$\begin{aligned} \mathrm{div}(L) &= (P) + (Q) + (S) - 3(\mathcal{O}) \\ \mathrm{div}(V) &= (R) + (S) - 2(\mathcal{O}). \end{aligned}$$

These two are both principal divisors on the elliptic curve, since they are the divisors of two line-functions. When we add in the quotient group Pic_E^0, we compute modulo a principal divisor. Thus, it holds that:

$$\begin{aligned} (P) + (Q) - 2(\mathcal{O}) &= ((P) + (Q) - 2(\mathcal{O})) \ominus \mathrm{div}(L) = \\ &= ((P) + (Q) - 2(\mathcal{O})) - ((P) + (Q) + (S) - 3(\mathcal{O})) = \\ &= -(S) + (\mathcal{O}) = (-(S) + (\mathcal{O})) \oplus \mathrm{div}(V) = \\ &= (R) - (\mathcal{O}). \end{aligned}$$

This formal approach towards addition will be of great further use in the explanation and computation of the Weil and Tate pairings, which are presented in the following section. My internship report, [27], contains more details about elliptic curves and their arithmetic. In the remainder of this thesis, we refer to elliptic curves over finite fields, both fields of prime order and extension fields, rather than general fields $\mathbb{K}$.

2.4 Pairings

Let G_1, G_2, and G_3 be groups of order l, where l is prime. Let G_1 and G_2 be groups under addition and G_3 be a group under multiplication.

Definition 2.4.1. *A* pairing *is a bilinear map of the form $e : G_1 \times G_2 \longrightarrow G_3$ with the properties ([6]):*

- ***Bilinear**: For $a, b \in \mathbb{Z}$ and elements $P \in G_1$ and $Q \in G_2$, it holds: $e(aP, bQ) = e(P, Q)^{ab}$. Note that in the left hand side, the first argument of the pairing involves scalar multiplication on G_1 and the second argument concerns scalar multiplication on G_2. The right hand side concerns exponentiation in G_3.*
- ***Nondegenerate**: Not all pairs $(P, Q) \in G_1 \times G_2$ are mapped to the identity element in G_3.*
- ***Computable**: There exists an efficient algorithm for the computation of this pairing.*

For cryptography, we require the notion of a cryptographic pairing.

Definition 2.4.2. *A* cryptographic pairing *is a pairing with the additional property:*

***Computable**: There exists an efficient algorithm for the computation of this pairing.*

In practice, pairings are constructed on elliptic or hyperelliptic curves. Not every curve can support an efficiently computable pairing; those that do are called pairing-friendly curves. We define such a curve after establishing the notion of an embedding degree.

Definition 2.4.3. *Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$; let l be a large prime dividing the number of points on the elliptic curve, i.e. $l | \#E(\mathbb{F}_q)$. The* embedding degree *with respect to l is the smallest integer k such that $l | q^k - 1$.*

Definition 2.4.4. *A non-supersingular elliptic curve over $\mathbb{F}_p$ is called pairing-friendly if it contains a subgroup of order l whose embedding degree k is not too large, which means that computations in the field $\mathbb{F}_{p^k}$ are feasible.*

In what follows we describe two of the most used types of pairings on elliptic curves: the Weil and the Tate pairings. Let q be a prime or a prime power and l a large prime such that $l | E(\mathbb{F}_q)$; let the subgroup of $\mathbb{F}_q$-rational points of order l on E be denoted G. Let k be the embedding degree of E with respect to l and let $k > 1$. By [3], all l-torsion points are defined over $\mathbb{F}_{q^k}$. Denote the set of $\mathbb{F}_{q^k}$-rational l-torsion points by $E(\mathbb{F}_{q^k})[l]$ and observe that it is a group under addition.

Definition 2.4.5. *The* Weil pairing *([6]) is a bilinear map defined by* $w : E(\mathbb{F}_q)[l] \times E(\mathbb{F}_{q^k})[l] \longrightarrow \mu_l$. *Here, the group* μ_l *represents the group of the* l^{th} *roots of unity of* $\mathbb{F}^*_{q^k}$. *The mapping is given by:*

$$w(P, Q) = \frac{f_P(\tilde{Q})}{f_Q(\tilde{P})} \,. \tag{2.11}$$

In this expression, $P \in E(\mathbb{F}_q)[l]$ *and* $Q \in E(\mathbb{F}_{q^k})[l]$. *The functions* f_P *and* f_Q *will be described in more detail below – they are functions on the elliptic curve.* $\tilde{Q}$ *and* $\tilde{P}$ *are divisors equivalent to* $(Q) - (\mathcal{O})$ *and* $(P) - (\mathcal{O})$*; they are chosen so that the functions* f_P *and* f_Q *are defined and non-zero on their support.*

In practice, the second argument of the pairing is a point in a subset of $E(\mathbb{F}_{q^k})$ of order l. It holds that $lP = \mathcal{O}$, as P is an l-torsion point. Remembering the equivalence of divisors (see the previous section), we can write in the divisor setting: $l((P) - (\mathcal{O})) \in \text{Princ}_E$. Thus, there is a function f_P depending on this point P such that $l((P) - (\mathcal{O})) = \text{div}(f_P)$. This is the function specified above, and it depends on the point P. The function f_Q is defined analogously.
This pairing gives as a result an l^{th} root of unity. However, the fact that (by construction) $w(P, P) = 1$ is a serious problem for some protocols assuming that $G_2 = G_1$. One way out is to generalise the protocol; this is done e.g. for BN curves (see chapter 3). Another way out is to introduce distortion maps from G_1 to G_2.

Definition 2.4.6. *A* distortion *map is an endomorphism* $\phi : E(\mathbb{F}_q) \longrightarrow E(\mathbb{F}_{q^k})$, *so that* $e(P, \phi(P)) \neq 1$.

While the Weil pairing is defined on any elliptic curve, usually it is not efficiently computable. It is generally quite difficult to find an elliptic curve, a corresponding embedding degree, and choose a pairing such that the setting is secure and efficiently computable. In applications therefore, the value of k should be small in order for the pairings to be computable, but not too small, as the larger k is, the smaller q needs to be for a cryptographic setting to be secure. Because of security considerations, however, q may also not become too small.
Boneh and Franklin illustrate the definition of distortion maps by considering the elliptic curve E defined over $\mathbb{F}_q$ by: $y^2 = x^3 + 1$. In this setting, a distortion map ϕ can be defined by $\phi(x, y) = (\zeta x, y)$, where $1 \neq \zeta \in \mathbb{F}_{q^2}$ is a solution of $x^3 - 1 = 0$. The parameters chosen by Boneh and Franklin in this example are not sufficiently good for secure communication, as the Decisional Diffie-Hellman problem (see chapter 4) is not hard enough in this setting – see [20].
Once we have a distortion map, we define the modified Weil pairing: $\hat{w} : E(\mathbb{F}_q)[l] \times E(\mathbb{F}_q)[l] \longrightarrow \mu_l$ with:

$$\hat{w}(P, Q) = w(P, \phi(Q)). \tag{2.12}$$

This new mapping preserves all the properties of the initial Weil pairing, but satisfies $\hat{w}(P, P) \neq 1$. This makes e.g. key exchange protocols work with the pairing.
The Tate pairing offers a faster computation than the Weil pairing.
Consider the bilinear, nondegenerate map: $\mathcal{T} : E(\mathbb{F}_q)[l] \times (E(\mathbb{F}_{q^k})/lE(\mathbb{F}_{q^k})) \longrightarrow \mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^l$ with:

$$\mathcal{T}(P, Q) = f_P(\tilde{Q}), \tag{2.13}$$

where f_P and $\tilde{Q}$ are defined as in the case of the Weil pairing. Let $k > 1$. We have that $(E(\mathbb{F}_{q^k})/lE(\mathbb{F}_{q^k})) \cong E(\mathbb{F}_{q^k})[l]$.

Definition 2.4.7. *The* modified Tate pairing *is defined as:* $\hat{\mathcal{T}} : E(\mathbb{F}_q)[l] \times (E(\mathbb{F}_{q^k})/lE(\mathbb{F}_{q^k})) \longrightarrow \mu_l$, *with:*

$$\hat{\mathcal{T}}(P, Q) = (\mathcal{T}(P, Q))^{\frac{q^k-1}{l}}. \tag{2.14}$$

The output of the pairing is indeed an l^{th} root of unity, as $((\mathcal{T}(P, Q))^{\frac{q^k-1}{l}})^l = (\mathcal{T}(P, Q))^{q^k-1} = 1$. We can use a distortion map as before to map a point on $E(\mathbb{F}_q)$ to a point in $E(\mathbb{F}_{q^k}) \backslash E(\mathbb{F}_q)$.
The computation needed for this modified Tate pairing is therefore about half the computation needed for the Weil pairing, with the extra exponentiation at the end.
Miller's algorithm computes f_P evaluated at a divisor; this result can be used to compute of the Tate and Weil pairings as defined above. As before, in the definition of the Weil pairing, $\tilde{Q} \equiv (Q) - (\mathcal{O})$. To choose an equivalent divisor, we pick a random S and put $(Q) - (\mathcal{O}) = (Q) + (S) - (S) - (\mathcal{O}) = (Q + S) - (S)$. The addition of Q and S is point addition, while $(Q)+(S)$ represents a formal addition of points, leading to a divisor. It holds that:

$$f_P(\tilde{Q}) = f_P((Q + S) - (S)) = \frac{f_P(Q + S)}{f_P(S)}. \tag{2.15}$$

The computation of f_P is done step by step, evaluating the argument in every step of the computation. We have as input $P \in E(\mathbb{F}_q)[l]$ and $Q \in E(\mathbb{F}_{q^k})[l]$, as well as an $m + 1$-bit prime l. We first write: $l = \sum_{i=0}^{m} l_i 2^i$, where $l_i \in \{0, 1\}$ and $l_m = 1$. The output will be the modified Tate pairing of P and Q, i.e. $\hat{\mathcal{T}}(P, Q)$. Write $R = Q + S$. The algorithm is:

Miller's Algorithm

1. $T \leftarrow P$; $f_1 \leftarrow 1$;
2. FOR $i = m-1, m-2, \ldots, 1, 0$ DO:
   (a) $f_1 \leftarrow f_1^2 \frac{l_{T,T}(R)\cdot v_{2T}(S)}{v_{2T}(R)\cdot l_{T,T}(S)}$;

   $T \leftarrow 2T$;

   (b) If $l_i = 1$ THEN: $f_1 \leftarrow f_1 \frac{l_{T,P}(R)\cdot v_{T+P}(S)}{v_{T+P}(R)\cdot l_{T,P}(S)}$;

   $T \leftarrow T + P$.
3. RETURN: $f_1^{\frac{q^k-1}{l}}$.

In this algorithm, $l_{T,P}$ denotes the equation of the line through T and P. The expression $l_{T,P}(R)$ denotes the evaluation of the line function through T and P in the point R. That means: in a line equation of the kind $y = mx + c$, we replace y by the y-coordinate of the point R and x with its x-coordinate. The notation v_P represents the vertical line through P, i.e. the line through P and $\mathcal{O}$. Similarly, $v_P(Q)$ is the evaluation of the vertical line equation through P at the point Q. By extension, $v_{T+P}(S)$ is the vertical line through $T+P$ evaluated at point S. The computation of the Weil pairing involves using Miller's algorithm to compute $f_P(Q)$ and $f_Q(P)$ without the final exponentiation.
There are certain speed-ups that can be used for this algorithm. A particular speed-up appears for k even. Choose $Q \in E(\mathbb{F}_{q^k})$ with $x_Q \in \mathbb{F}_{q^{\frac{k}{2}}}$ and $y_Q \in \mathbb{F}_{q^k}$ and not defined over a smaller field. Clearly, the vertical line through the point $2T$ and the vertical line through the point $T+P$ involve x and not y, and so the evaluation of these functions in Q gives a value in $\mathbb{F}_{q^{\frac{k}{2}}}$. We have that $v_{2T}(Q) = x_Q - x_{2T}$. Thus, $v_{2T}(Q)^{\frac{q^k-1}{l}} = (x_Q - x_{2T})^{\frac{q^k-1}{l}}$. We denote $x_Q - x_{2T} = \alpha$. Since $x_Q \in \mathbb{F}_{q^{\frac{k}{2}}}$ and $x_{2T} \in \mathbb{F}_q$, it holds that $\alpha \in \mathbb{F}_{q^{\frac{k}{2}}}$.
Since k is the embedding degree and thus chosen minimal so that $l | q^k - 1$, l does not divide $q^{\frac{k}{2}} - 1$. Therefore, the map $\varphi : x \longrightarrow x^l$ is a bijection in $\mathbb{F}_{q^{\frac{k}{2}}}$. There exists therefore a $\beta \in \mathbb{F}_{q^{\frac{k}{2}}}$ such that:

$$\beta = \varphi^{-1}(\alpha). \tag{2.16}$$

This translates to: $\alpha = \varphi(\beta) = \beta^l$. The pairing algorithm ends by raising the result to the power $\frac{q^k-1}{l}$. All the contributions to f_1 are multiplicative and the exponentiation can thus be applied to each factor separately. Therefore:

$$(v_{2T}(Q))^{\frac{q^k-1}{l}} = \alpha^{\frac{q^k-1}{l}} = (\beta^l)^{\frac{q^k-1}{l}} = \beta^{(q^k-1)} = 1. \tag{2.17}$$

The same argument can be used for the second vertical line, $v_{P+T}(Q)$. The point S can be chosen in $E(\mathbb{F}_q)$ itself so that all contributions of S are annihilated. Therefore, both division steps become obsolete for Tate pairings for which the embedding degree is even. Similarly, all computations involving only S in Miller's algorithm can be eliminated as soon as $k > 1$. In elliptic curve cryptography, pairings can be used to improve speed, but their existence has a few consequences. One such consequence is that one of the hard problems in elliptic curve cryptography becomes easily solvable. More details regarding hard problems for pairing based cryptography can be found in chapter 4.

Chapter 3

Various Cryptographic Settings

Throughout the course of this thesis, we will often refer to the notion of a cryptographic setting. This term is only used in this thesis, and so we define it locally as an underlying cryptographic primitive – for example a field – and the arithmetic associated with it, which permits secure communication by means of several pre-established algorithms.

There are several cryptographic settings that are already in use. Most of the protocols that can perform cryptography in these settings can be translated from one setting to another; most such protocols can also be abstractly stated in a cyclic group.

3.1 The Finite Fields Setting

Let p be a prime number and denote $\mathbb{F}_p$ the finite field of integers modulo p under addition and multiplication. Let l be a prime such that $l|p-1$. The identity element of addition is in the equivalence class of 0, while the identity element of multiplication is the equivalence class of 1. We consider the multiplicative group of $\mathbb{F}_p$, i.e. the set $\mathbb{F}_p^* = \mathbb{F}_p \setminus \{0\}$. We consider an element $g \in \mathbb{F}_p^*$ such that g has prime multiplicative order l.

The fact that g has prime order l means that $g^1, \ldots, g^{l-1}$ are all different and not equivalent to $g^0 = 1$ (if any of them were equivalent to 1, then there would exist an integer $k < l$ such that $g^k = 1$; this cannot happen as l is the order of g). For the sake of clarity, we mention that whenever calculations are made in $\mathbb{F}_p$, the result is automatically reduced modulo p. Therefore, we will use the notation $a = b$ for $a \equiv b \mod p$. Clearly, under multiplication modulo p, g generates a finite cyclic subgroup of order l within $\mathbb{F}_p^*$. Key exchange can be performed in this subgroup.

The parameters p, l, and g are public in all key-exchange protocols. The final section of this chapter and the next chapter add a few considerations on the size of these parameters; we denote the bit size of p by s_p and the size of l by s_l. The method to generate these parameters is to begin by finding a suitable s_l-bit prime l.

At this point, we state Fermat's little theorem:

Theorem 3.1.1. *(**Fermat's Little Theorem:**) Let p be a prime number. Then for any integer g, the number $g^p - g$ is divisible by p.*

A proof of Fermat's little theorem can be found in [2].

This theorem can be reformulated in finite field arithmetic as follows: for any element $g \in \mathbb{F}_p^*$ it holds that $g^p = g$, or equivalently $g^{p-1} = 1$.

Thus it holds that $l|(p-1)$. We can write: $p - 1 = lm$. Having already found a prime l of size s_l, we now randomly pick an integer m of $s_p - s_l$ bits and check whether $lm + 1$ is prime. If it is, then we set $p = lm + 1$. If it isn't, then we choose another m and repeat the procedure until a suitable p has been found. All that is left now is to find an element $g \in \mathbb{F}_p$ of order l. This can be done as follows: pick a random element a and check if $a^k = 1$. If that doesn't hold, write $g = a^m$. If $a^m = 1$, then pick another a and start over again.

The fact that a^m has order l if $a^m \neq 1$ is not difficult to see. By Fermat's little theorem, it holds that $a^{p-1} = 1$, so $a^{lm} = 1$. Therefore $(a^m)^l = 1$. l is prime, therefore the order of a^m must be l. Together, the two primes p and l, and the element g form a finite field setting for key exchange protocols; they are generally called system parameters.

3.2 The Elliptic Curve Setting

Elliptic curves offer another cyclic group for cryptography to be applied upon. A few notions on elliptic curves are described in chapter 2.

We recall that any curve over a field of characteristic greater than 3 is isomorphic to one in short Weierstrass form:

$$y^2 = x^3 + ax + b \ . \tag{3.1}$$

In this thesis, we work with curves of large characteristics, therefore the curves we consider will have the short Weierstrass form and can be determined by two integers, $a, b \in \mathbb{F}_q$.

Let E be an elliptic curve defined over $\mathbb{F}_q$ and let l be a prime such that $l|\#E(\mathbb{F}_q)$ and $l^2 \nmid \#E(\mathbb{F}_q)$. Consider an l-torsion $\mathbb{F}_q$-rational point on E and denote this point P. Similar to the case of integers modulo q, we have that P generates a finite cyclic subgroup of l-torsion points of cardinality l. We denote the point at infinity on E by $\mathcal{O}$; this point is part of the subgroup $E(\mathbb{F}_q)[l]$ generated by P. We denote point addition on E by $+$. Repeated addition on the elliptic curve is called scalar multiplication.

The elements of the cyclic subgroup of l-torsion $\mathbb{F}_q$-rational points of E generated by P are given by $\{\mathcal{O}, P, ...(l-1)P\}$. These are all distinct points, and $lP = \mathcal{O}$. It is within this cyclic group that cryptography can be used.

Choosing the parameters, i.e. l and q, depends on the desired level of security, as in the case of finite fields. More details on hard problems on elliptic curves follow in the next chapter, but it suffices to say that usually the parameters in elliptic curve cryptography are much smaller in size than in the finite field case. That is where the advantage of using elliptic curve cryptography lies.

Choosing parameters for an elliptic curve is not an easy task. For cryptography on elliptic curves, the sizes of the integers q and l, which we denote s_q and s_l respectively, are comparable. NIST and SECG have both presented documentation relating recommended choices

for the domain parameters in elliptic curve cryptography. These documents provide suitable q, l, E, and $m = \frac{q-1}{l}$.

It still remains to find a point $P \neq \mathcal{O}$ of prime order l. Similar to the finite field case, we simply pick a random point Q on the elliptic curve and check if $mQ = \mathcal{O}$. If so, we repeat the procedure until we find Q for which $mQ \neq \mathcal{O}$. We set $P = mQ$ and have the point of prime order l on E. The point P now generates a finite cyclic subgroup of order l in $E(\mathbb{F}_q)$.

A cryptographic setting for elliptic curve based key exchange includes the two primes q and l, the elliptic curve E defined on $\mathbb{F}_q$, and a point P of order l generating a cyclic subgroup on $E(\mathbb{F}_q)$.

3.3 The Pairing Setting

Some details on pairings have already been presented in chapter 2. We now add a few practical considerations related to the pairing setting. First, we consider the embedding degree. Let $\mathbb{F}_q$ be a finite field, with q prime or a prime power, and consider an elliptic curve E over $\mathbb{F}_q$. Let l be a large prime with $l|\#E(\mathbb{F}_q)$. At this point, we state without further explanations that the choice of the embedding degree is a trade-off between making the parameters (mainly q) as small as possible and still keeping the pairing computable and the DLP in $E(\mathbb{F}_q)$ and $E(\mathbb{F}_{q^k})$ (see chapter 4) secure. Implementations of key exchange protocols in pairing settings exist already for various values of k, such as for $k = 2$, $k = 6$, $k = 10$, and $k = 12$.

For some of these values of k, pairings can be considered with distortion maps; in other cases, distortion maps do not exist and so the protocols are modified. As we have seen in chapter 2, while $k > 1$, both the Weil and Tate pairings are unsuitable for cryptography if they take their arguments from the same group, for example from the subgroup of l-torsion points generated by a point P. The distortion map modifies the second argument of the pairing by applying a transformation on P; if distortion maps do not exist, the solution is to choose the second argument from a subgroup of l-torsion points generated by a point Q not in the subgroup of l-torsion points generated by P.

The parameters for key exchange in a pairing setting therefore are: q, l, P as in the elliptic curve setting, to which one additionally needs the embedding degree k. If a distortion map exists, the distortion map ϕ is part of the setting; if it does not exist, then the point Q is a part of the setting. It is possible that additional conditions on q are necessary so as to suit the choice of the distortion map.

In what follows, we give a brief overview of some choices of elliptic curves for the smallest non-trivial embedding degree, $k = 2$, and for the BN-curves – which are some of the more efficient pairing environments and have $k = 12$.

3.3.1 The Case for $k = 2$

The choice of pairing and distortion map depends very much on the choice of elliptic curve. For the case of $k = 2$, the parameters of the pairing setting need still be relatively large, with

q of 512 bits and l of 160 bits providing minimal security. Boneh and Franklin ([6]) give a few examples of suitable pairing settings for an embedding degree of 2. We give here a brief overview of two such settings and refer the reader to the original paper for more details.

We first consider the case when q is a prime with $q = 2 \mod 3$. We choose the elliptic curve E with equation $y^2 = x^3 + 1$ on $\mathbb{F}_q$. This curve has $q + 1$ points. The condition that $q = 2 \mod 3$ ensures that $q^2 = 1 \mod 3$ and such that $\mathbb{F}_{q^2}$ contains a third root of unity, which $\mathbb{F}_q$ does not.

In this environment, we consider a pairing e – which could be for example the Tate pairing – and a distortion map given by $\phi(x, y) = (\zeta x, y)$, for $1 \neq \zeta \in \mathbb{F}_{q^2}$. The modified Tate pairing for this distortion map is $\hat{e}$ defined with $\hat{e}(P, Q) = \hat{e}(P, \phi(Q))$.

This choice of parameters, though valid, is slow. This is a consequence of the fact that the distortion map ϕ maps the $x-$coordinate of the point in the extension field $\mathbb{F}_{q^2}$. Therefore, the computational speed-ups in 2 do not apply.

A choice which does allow for all the shortcuts described in the previous chapter has also been given by Boneh and Franklin. This setting has been used for the implementation of pairing-based BD I in [27]. We take q to be a prime with the property that $q = 3 \mod 4$. This property will ensure that $q^2 = 1 \mod 4$, so that $\mathbb{F}_{q^2}$ contains a fourth root of unity $\zeta \notin \mathbb{F}_q$.

Over $\mathbb{F}_q$ we now define the elliptic curve E by the equation: $y^2 = x^3 + x$. We take a pairing e and modify it to $\hat{e}$ with the aid of a distortion map ϕ with $\phi(x, y) = (-x, \zeta y)$, such that $\zeta^2 + 1 = 0$ in $\mathbb{F}_{q^2}$. We note that ζ is equivalent to the complex i. In this case, since the coordinate x was already in $\mathbb{F}_q$, its negative will also be an element of the same field, and Miller's algorithm will be computed without inversions.

Once more, the choosing of the parameters begins with finding the prime order l. We denote the sizes of l and q s_l and s_q respectively. The paper [6] provides a few examples of elliptic curves that can be used for cryptography; these are not always secure, however. Finding an elliptic curve that is suitable for performing key exchange is quite a difficult task; a simple example that should nevertheless not be used for secure applications is $E : \quad y^2 = x^3 + x$. The number of points on E is $\#E(\mathbb{F}_q) = q + 1$, so we are looking for a prime q such that $l | \#E(\mathbb{F}_q)$ and $l^2 \nmid \#E(\mathbb{F}_q)$. We write, as in section 3.1, $q + 1 = lm$ for some integer m. Once a suitable prime l has been found, one repeatedly searches for an integer m of size $s_q - s_l$ so as to ensure that the number $lm - 1$ is prime. Once such an m is found, one sets $q = lm - 1$. Subsequently finding a point P of prime order l is done by the same method as shown in the previous section.

An example of suitably chosen parameters q, l, and $P \in E(\mathbb{F}_q)$ of prime order l can be found in [27].

3.3.2 The Case for $k = 12$

The main advantage of elliptic curves is that they can provide a high level of security with relatively small-sized parameters. This advantage can also be provided by pairings, as the security of cryptographic protocols on pairing settings depends on bilinear hard problems in G_1 and on finite fields hard problems in G_3 (see chapter 4 for details on hard problems). The

higher the embedding degree, the smaller the value of q; therefore an embedding degree of 12 puts pairings to much greater advantage than $k = 2$. Paulo Barreto and Michael Naehrig have proposed a method of finding pairing-friendly elliptic curves in [4].

Let E be a pairing-friendly BN curve and let the setting parameters q and l be chosen such that the curve has an embedding degree $k = 12$. BN curves do not admit distortion maps. Therefore, pairing-based cryptographic protocols must be modified for this case in order to make sure that $e(P, P) \neq 1$ for a point P on the elliptic curve and for the pairing e.

In their work, the authors of [4] provide improvements for computations in cryptographic applications, namely what they name point compression and pairing compression. Without going into further details, we mention that by means of a sextic twist, Barreto and Naehrig are able to write points in $E(\mathbb{F}_{q^{12}})$ as points in $E'(\mathbb{F}_{q^2})$, where the elliptic curve E' is obtained by twisting E. The same trick allows for the use of only $\mathbb{F}_q$ and $\mathbb{F}_{q^2}$ arithmetic for non-pairing operations – such as for example key generation.

[4] further optimises the cryptography on these curves by representing points on the twisted elliptic curve E' by means of their y coordinate only. Some concrete examples are given at the end of [4], based on an elliptic curve of the form $y^2 = x^3 + 3$. The primes q and l are explicitly given, with q chosen such that the computation of cubic roots is easy (this step is required for pairing compressions and the representation of points by their y coordinate only). For q of 160 bits, we already obtain a level of security comparable to a finite field setting where p is chosen to have 1820 bits.

3.4 An Overview of these Three Settings

The three settings presented above appear rather different. Nevertheless, they allow the translation, from one to the other, of several established cryptographic key exchange protocols. We give a comparative overview of the three settings below. The sizes specified at the bottom of the table are those recommended for an 80-bit security (equivalent to choosing p of 1024 bits in a finite field setting). We relate a level of security of b bits to parameter sizes that ensure that the best known attack against the hard problems (see chapter 4) in that setting is of time complexity of order 2^b. In the elliptic curve setting, the Pollard Rho method (presented in chapter 4) is the best known attack, and therefore the elliptic curve setting demands $2b$ bits of security. On finite fields, the best known attack is the Index Calculus method, so the parameters asymptotically converge to b^3 bits.

Finite Fields	Elliptic Curves	Pairings
$\mathbb{F}_p$	$E(\mathbb{F}_q)$	$E(\mathbb{F}_q)$
$l \mid p-1$	$l \mid \#E(\mathbb{F}_q)$	$l \mid \#E(\mathbb{F}_q)$
$g \in \mathbb{F}_p$	$P \in E(\mathbb{F}_q)$	$P \in E(\mathbb{F}_q)$
–	–	k
–	–	ϕ
integers mod p	points on $E(\mathbb{F}_q)$	point pairs on $E(\mathbb{F}_q)$
multiplication mod p	point addition on $E(\mathbb{F}_q)$	pairings on $E(\mathbb{F}_q)$
$s_p = 1024$	$s_q = 160$	$s_q \geq 160$
$s_l = 160$	$s_l = 160$	$s_l = 160$

As a further condition in the pairing setting, it must hold that $ks_q = 1024$ bits.

As the table shows, several parallels may be drawn between the three different settings presented already. It is important to note that a natural correspondence exists between the elements present in the three settings: integers modulo p, points on an elliptic curve E, and pairs of points on an elliptic curve. Each type of element is related to a natural operation defined on its respective setting.

We note that multiplication and exponentiation modulo p are very simple and efficient operations, which require very little time and computation power. Point addition on elliptic curves is somewhat more labourious, requiring several finite field multiplications. Finally, pairing computations are even more complicated, requiring several point additions. Therefore, it would appear that the latter settings are less efficient than finite fields.

However, there are two points in favour of using pairings rather than finite fields. The first is that applications in the pairing setting generally try to contain as few pairing computations as possible, therefore much of the arithmetic involved in a cryptographic key exchange protocol in pairing settings is made up of simple finite field operations.

Secondly, the nature of the pairing and elliptic curve settings allow for smaller parameters (for reasons explained in the following chapter). Therefore, although the individual operations might be simpler in finite fields, they are performed in larger fields, and are therefore more time consuming. We also note that, while there are several natural correspondences between the three settings, there are several fundamental differences in their structures.

Chapter 4

Hard Problems in Various Settings

Information theoretic security for cryptographic key exchange protocols is difficult to achieve. Quantum cryptography attempts to achieve this by means of detecting eavesdropping and aborting the connection if too much interference is detected. In this way, information theoretic security can be achieved. For the protocols and settings presented in this paper, security is measured in the time and computation power it takes for an attacker to break the system; breaking the system generally boils down to solving one or more hard problems. In the remainder of this chapter, we give some of the important hard problems that exist in the various settings presented in chapter 3 and two of the more important methods that solve these problems.

4.1 Hard Problems in Finite Fields

The setting is as follows. We have prime numbers p and l such that $l|(p-1)$, and an element $g \in \mathbb{F}_p$ such that the order of g is l.

- **DLP** – Discrete Logarithm Problem – Given g and $g^x \in \mathbb{F}_p$, with $x \in \{2, \ldots, l-1\}$, find x.

- **(C)DHP** – (Computational) Diffie-Hellman Problem – Given g, g^x, and g^y, with $x, y \in \{2, \ldots, l-1\}$, find g^{xy}.

- **DDHP** – Decisional DHP – Given g, g^x, g^y, and g^z decide whether $g^{xy} = g^z$.

The Diffie-Hellman problem is referred to as the computational Diffie-Hellman problem (CDHP) as soon as one needs to consider the DDHP. This is done so as to distinguish between the two versions of the problem.

It is obvious that solving the discrete logarithm problem will solve the other two hard problems as well. It is not sure, however, that the only way to solve the CDHP and DDHP is via solving the DLP. There are a few methods to solve the DLP in finite fields, the most notable of which are the Pollard Rho and the Index Calculus methods. In the following section, we present these methods in short.

4.2 The Pollard Rho and Index Calculus Methods

We consider the setting of the previous section. We have primes p and l, such that $l|p-1$. We also have an element g of prime order l, such that g generates a cyclic subgroup $G \subset \mathbb{F}_p^*$. Given now $h \in G$, $h \notin \{0, 1\}$, we wish to find x such that $h = g^x$.

The fastest method to solve the DLP in finite fields is the Index Calculus method. The second fastest solution is given by the Pollard Rho method, and the reason we present this method here is that in the elliptic curve setting, this is the fastest method. The details of these methods as well as other algorithms to solve the DLP are given in [28].

4.2.1 The Pollard Rho Method

We consider the DLP as given in the introductory paragraph of section 4.2. The multiplicative subgroup G is now partitioned in three subsets, G_1, G_2, G_3 as determined by the following equivalence relation:

$$x \in G_i \quad \Leftrightarrow x \equiv i \ (\mathrm{mod}\ 3). \tag{4.1}$$

We recall that the existence of a partition means that the resulting subsets are disjoint, i.e. that they have no elements in common. Having this partition, we now define a recursive sequence $\{x_i\}_{i\geq 0}$ recursively, with $x_0 = g$ and $x_{i+1} = f(x_i)$:

$$\begin{aligned} f(x_i) &= x_i^2 \mod p \quad \text{if } x_i \in G_1 \\ f(x_i) &= hx_i \mod p \quad \text{if } x_i \in G_2 \\ f(x_i) &= gx_i \mod p \quad \text{if } x_i \in G_3 \end{aligned}$$

Since we are in a finite field, the sequence x_i will eventually begin to cycle back and yield the same function value for an x dependent only on g. The characteristic shape of this cycle – as shown below – resembles the Greek symbol Rho; hence the method was named Pollard Rho, after its inventor, Pollard.

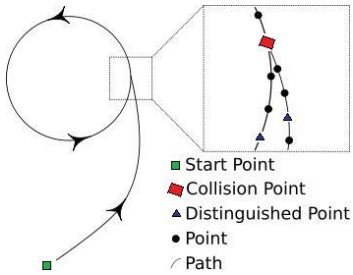

Figure 4.1: Pollard's Rho

We now create two additional sequences $\{a_i\}_{i\geq 0}$ and $\{b_i\}_{i\geq 0}$ such that $x_i = g^{a_i}h^{b_i}$ for any i. We must take a_0 and b_0 with $a_0 = 1$ and $b_0 = 0$ and then construct the sequences recursively as follows:

$$\begin{aligned} a_{i+1} &= 2a_i \mod l \quad \text{and} \quad b_{i+1} = 2b_i \mod l \quad \text{if } x_i \in G_1 \\ a_{i+1} &= a_i \mod l \quad \text{and} \quad b_{i+1} = b_i + 1 \mod l \quad \text{if } x_i \in G_2 \\ a_{i+1} &= a_i + 1 \mod l \quad \text{and} \quad b_{i+1} = b_i \mod l \quad \text{if } x_i \in G_3 \end{aligned}$$

It can be proven that indeed $x_i = g^{a_i}h^{b_i}$. Take first the case $x_i \in G_1$. Then $x_{i+1} = x_i^2 = (g^{a_i}h^{b_i})^2 = g^{2a_i}h^{2b_i} = g^{a_{i+1}}h^{b_{i+1}}$. Then take $x_i \in G_2$. It follows that $x_{i+1} = hx_i = hg^{a_i}h^{b_i} = g^{a_i}h^{b_i+1} = g^{a_{i+1}}h^{b_{i+1}}$. Finally, if $x_i \in G_3$, we have that: $x_{i+1} = gx_i = gg^{a_i}h^{b_i} = g^{a_i+1}h^{b_i} = g^{a_{i+1}}h^{b_{i+1}}$. This is an inductive proof, and the basis of the induction is that $x_0 = g = g^1h^0$, with $a_0 = 1$ and $b_0 = 0$.

The main idea of the Pollard Rho method is now to find indices $i \neq j$ such that $x_i = x_j$. This would mean $g^{a_i}h^{b_i} = g^{a_j}h^{b_j}$. Thus, $g^{a_i-a_j} = h^{b_j-b_i}$ and thus, since $h = g^x$ for some unknown x, $g^{a_i-a_j} = g^{x(b_j-b_i)}$ and $x \equiv \frac{a_j-a_i}{b_i-b_j} \mod l$.

Although it is quite unlikely, it might be the case that $b_i = b_j$. If that happens, we write $h' = g \cdot h$ and note that $h' = g^{x+1}$. In this case we now solve $h' = g^{x'}$, for $x' = x + 1$. Having found x', we find x.

In order to find i and j such that $x_i = x_j$, we use a method called Floyd's cycle-finding algorithm (see [22]), which attempts to find an index i such that $x_i = x_{2i}$. We start with the pair (x_1, x_2) and then calculate (x_{i+1}, x_{2i+2}) from (x_i, x_{2i}) by the general rule: $x_{i+1} = f(x_i)$ and $x_{2i+2} = f(f(x_{2i}))$. This algorithm minimises the storage requirements for the Pollard Rho method. The computations necessary to find the secret index x are expected to be finished in $O(\sqrt{p})$.

4.2.2 The Index Calculus Method

Once more, we consider a finite field setting, with p, l primes such that $l|p-1$. We have an element $g \in \mathbb{F}_p$ generating a cyclic, multiplicative subgroup G of prime order l, containing

the elements $\{1, g, \ldots, g^{l-1}\}$. Given $h \in G$, we must find $x \in \{2, \ldots, l-1\}$ such that $g^x = h$.

As stated before, the elements of $\mathbb{F}_p$ and $\mathbb{F}_p^*$ are represented by the smallest positive representative in their equivalence classes. In what follows, these representatives are them embedded in $\mathbb{Z}$, thus obtaining integers.

The Index Calculus method proceeds then according to the following basic steps:

1. We first select an appropriate $S \subset \{1, \ldots p-1\}$ such that many elements of $\{1 \ldots p-1\}$ can be efficiently written as a product of elements of S. We call S a *factor base*. We call an element that can be efficiently written in terms of the elements in S is called *smooth* with respect to S. We denote $|S| = n$, for some n. The natural choice for the set S would be for instance a set of primes in $\{1, \ldots p-1\}$. Our goal now is to find the discrete logarithm of each $\overline{s} = g^{a_s} \in \mathbb{F}_p$ such that $\overline{s}$ is represented by an $s \in \mathbb{F}_p$.

2. Find a set I of exponents i such that for all $i \in I$, g^i can be efficiently expressed in terms of elements of S. We will then have a set of equalities of the form:

$$g^i = s_1^{u_{i,1}} s_2^{u_{i,2}} \ldots s_n^{u_{i,n}}. \tag{4.2}$$

   If we take the discrete logarithm to the base g on both sides, we will have, for all $i \in I$, a set of linear equations of the form:

$$i \equiv u_{i,1} \log_g(s_1) + u_{i,2} \log_g(s_2) + \cdots + u_{i,n} \log_g(s_n) \mod l. \tag{4.3}$$

   In these equations, i and the exponents $u_{i,1}, \ldots, u_{i,n}$ are known for every i. We treat $\log_g(s_j)$ for $j \in \{1, 2, \ldots, n\}$ as unknowns. If we have at least n independent equations, then we can solve the system of linear equations.

3. We solve the system of linear congruence equations in the unknowns $\log_g(s_j)$, for $1 \leq j \leq n$. At this moment we can write all the elements s_j in the form $s_j = g^{x_j}$.

   We have, at the moment, solved the DLP for the elements in the factor base S.

4. We now pick a random $r \in \{1, \ldots, l-1\}$ and try to write $g^r h$ as a product of elements in S. If that succeeds, then $g^r h = g^{r+x} = s_1^{v_1} s_2^{v_2} \ldots s_n^{v_n}$ and by taking the g-logarithm on both sides, we can write:

$$r + x = v_1 \log_g(s_1) + \cdots + v_n \log_g(s_n). \tag{4.4}$$

   We know all the g-logarithms of the elements of S, and we have chosen r ourselves, thus the only unknown remains x. Clearly, the size of S should be large enough for a random element $g^r h \in G$ to be likely to be written in terms of elements in S. On the

other hand, the size of S needs to be small, for the sake of storage and so as to be able to compute step 3.

In the situation described at the beginning of this section, the easiest choice of a factor base is to include the first n prime numbers. For a sufficiently large n, we will have enough elements easily expressed in terms of the elements of S. To write an element in G in terms of the elements of S, we need to successively divide the element by the various primes.

The complexity of this method is $\exp^{C\sqrt{\ln(p)\ \ln(\ln(p))}}$. This is a subexponential complexity, and in general will yield the result quicker than the Pollard Rho method.

4.2.3 Consequences of the two Methods

As stated before, the Pollard Rho algorithm is slower than the Index Calculus method in finding the exponent x. In fact, the Pollard Rho method runs in exponential time in the group order, while the Index Calculus method reaches a solution in subexponential time in the field size. However, while it is relatively easy to translate the Pollard Rho approach from a finite field to an elliptic curve setting (by exchanging integers modulo p and their multiplication by elliptic curve points and their addition), there are a few tricky points in translating the Index Calculus method, namely:

- How to effectively choose S.
- How to write elements of G in terms of elements of S.

In general, there are few environments that permit an easy and logical choice of S. Finite fields are one such structure, but elliptic curves are not. We generally pick a point P and generate the subgroup $E(\mathbb{F}_q)[l]$; though in fact all the points in this subgroup – apart from the point at infinity – are generators, in the remainder of this thesis, we generally refer to P as "the" generator of $E(\mathbb{F}_q)[l]$. Apart from P, there are no special points that could serve as a factor base. Additionally, even once those points were found, it is hard to find a way of writing the elements of the subgroup in terms of the components of the factor base. The direct consequence of this is that, while finite field cryptography has to be able to resist Index Calculus attacks, elliptic curve cryptography only has to withstand Pollard Rho attacks, which are less effective and have higher complexity.

It follows that the length of the finite field parameter p has to be much greater than the corresponding elliptic curve parameter q. The size ratio is given, without further explanation, in section 3.4. The individual finite field operations therefore are more efficient in the elliptic curve setting than in finite fields.

4.3 Hard Problems on Elliptic Curves

The setting is as follows. We have an elliptic curve E defined on a finite field $\mathbb{F}_q$, for q a prime or a prime power. Let l be a prime such that $l|\#E(\mathbb{F}_q)$, and $P \neq \mathcal{O}$ an l-torsion point

on the elliptic curve (as l is prime, this means that l is the order of P). Let G be the cyclic subgroup of $E(\mathbb{F}_q)[l]$ generated by P. The hard problems in the elliptic curve setting are as follows:

- **ECDLP** – Elliptic curve DLP – Given P and $Q = kP$ in G, find k.
- **ECCDHP** – Elliptic curve CDHP – Given P, $Q = aP$, $R = bP$ in G, find abP.
- **ECDDHP** – Elliptic curve DDHP – given P, $Q = aP$, $R = bP$, $S = cP$ in G, decide whether $c \equiv ab \pmod{l}$.

On elliptic curves, the solution of the ECDLP is most efficiently solved by the Pollard Rho method. As it is not possible to solve the discrete logarithm problem with the aid of the Index Calculus method on elliptic curves, the size of the parameters may be much smaller. According to [13], the key size needs to be 160 bits long for 80-bit security on the elliptic curve.

4.4 Bilinear Hard Problems

While pairings are theoretically achievable on all elliptic curves, usually they are not efficiently computable, as explained in section 3.2. We consider here a pairing defined as $e : G_1 \times G_2 \longrightarrow G_3$; this pairing is generally based on the Weil and Tate pairings. The existence of computable pairings has two main security-related implications. Firstly, the discrete logarithm problem in the group G_1 is no harder than the discrete logarithm problem in the group G_3. This has been proved in [26]. Therefore, in order to make this map secure with respect to the discrete logarithm problem, one has to choose the groups G_1 and G_3 such that both the security degrees are acceptable.

We consider the elliptic curve setting with E an elliptic curve over $\mathbb{F}_q$, for q a prime or a prime power. Furthermore, we consider a point $P \in E(\mathbb{F}_q)$ of prime order $l | \#E(\mathbb{F}_q)$; let furthermore $G_1 :=< P >$, the subgroup of $E(\mathbb{F}_q)$ generated by P. We consider a given, acceptably-large embedding degree k. We consider an efficiently computable, bilinear, non-degenerate map $e : G_1 \times G_2 \longrightarrow G_3$, with G_3 the subset of l^{th} roots of unity and G_2 another group of points such that a map exists between G_1 and G_2 and it is known. We denote the second argument of the pairing by $\hat{Q}$ rather than Q.

In this setting, the ECDDHP is easily solvable when efficient pairing computation exists. Indeed, for the ECDDHP, given aP and bP, one can calculate $e(aP, b\hat{P}) = e(P, \hat{P})^{ab}$. We can also calculate: $e(P, c\hat{P}) = e(P, \hat{P})^{c}$. If the two values are equal, clearly $c \equiv ab \mod l$.

For settings such as these, one can base cryptographic protocols on the bilinear versions of the hard problems that exist on finite fields and elliptic curves. These bilinear hard problems are shown below:

- **DLP** – ECDLP – given P and $Q = aP$, find a.

- **BCDHP** – Bilinear CDHP – given P, $Q = aP$, $R = bP$, $S = cP$ in G, find $e(P, \hat{P})^{abc}$.

- **BDDHP** – Bilinear DDHP – given P, $Q = aP$, $R = bP$, $S = cP$ in G, and $e(P, \hat{P})^{w}$, decide if $abc \equiv w \mod l$.

In the pairing setting, several types of attacks are possible. While each pairing computation owes its security degree to the bilinear hard problems, the results of pairings and any subsequent arithmetic performed on them are elements of the group G_3, and they rely on the hardness of the DLP in G_3, just as well as on that in G_1 and G_2.

4.5 Overview of Hard Problems

The hard problems in the three different settings are relatively similar to each other. An easy correspondence can be made between them, when one considers the particularities of their environments, as presented in chapter 3. We attempt to give the reader a feeling of the similarities by placing the three settings head to head, in the table below. To eliminate superfluous wording, we introduce the notation $a \rightsquigarrow b$ for the text: "Given a, find b."

Finite Fields	Elliptic Curves	Pairings
Parameters		
$\mathbb{F}_p$	$E(\mathbb{F}_q)$	$E(\mathbb{F}_q)$
$l \mid p-1$	$l \mid \#E(\mathbb{F}_q)$	$l \mid \#E(\mathbb{F}_q)$
$g \; in \mathbb{F}_p$	$P \in E(\mathbb{F}_q)$	$P \in E(\mathbb{F}_q)$
–	–	k
–	–	ϕ
–	–	e
Hard Problems		
$g, g^x \rightsquigarrow x$	$P, mP \rightsquigarrow m$	$P, mP \rightsquigarrow m$
$g, g^x, g^y \rightsquigarrow g^{xy}$	$P, aP, bP \rightsquigarrow abP$	$P, aP, bP, cP \rightsquigarrow e(P, \hat{P})^{abc}$
$g, g^x, g^y, g^z \rightsquigarrow$? $xy \equiv z \mod l$	$P, aP, bP, cP \rightsquigarrow$? $c \equiv ab \mod l$	$P, aP, bP, cP, e(P, \hat{P})^{w} \rightsquigarrow$? $w \equiv abc \mod l$

It is visible from this overview that all the hard problems specific to the pairing setting have an additional known variable, due to the fact that pairings take two parameters rather

than the simple, linear operations in finite fields and on elliptic curves, which only take one parameter.

A number of cryptographic protocols, such as bipartite and tripartite key exchange, and multipartite key exchange, rely on the computational hardness of these problems. We present a few such protocols in the following chapter.

Chapter 5

Protocols in Various Settings

Under the security assumptions from chapter 4, key exchange can now take place in any of the settings presented in chapter 3. The general setting of the protocol is the following: we have n parties wishing to securely communicate together, preferably with a single, common key. An initial, particular instance of key exchange for a small number of participants can be built in each setting, which can then be used to develop a general protocol for larger values of n. We will show that the smallest unit of key exchange in finite fields and in a basic elliptic curve setting is a line – two users – while with pairings, the basic unit is tripartite key exchange, arranged in a triangle shape. The two very-most basic protocols that achieve this minimal-user key exchange are Diffie-Hellman and Joux, respectively.

In section 5.1, we show basic key exchange for two and three users, respectively. These protocols will be further generalised to n users in section 5.2.

5.1 Basic Key Exchange in Various Settings

5.1.1 The Diffie-Hellman Key Exchange

In the finite field setting and on elliptic curves, we consider the case for two users, Alice and Bob, in short A and B. They can communicate with each other over an insecure channel, which can be eavesdropped by a third party, Eve. We consider Eve to be a passive attacker, i.e. she can intercept any message between A and B, but she cannot intervene in the communication. In what follows, we will show a protocol due to Diffie and Hellman ([12]) which ensures secure key exchange for Alice and Bob.

We first consider the finite fields key exchange setting. Let A and B both agree on a public set of finite fields parameters: p, l, and g, such that p and l are primes, with $l|p-1$. We consider G to be the cyclic subgroup of $\mathbb{F}_p$ generated by an element g of order l.

Alice		Bob
knows p, l, g		knows p, l, g

Step One

chooses $s_A \in \{2, \ldots, l-1\}$		chooses $s_B \in \{2, \ldots, l-1\}$
computes: $p_A = g^{s_A}$	$\longrightarrow$	receives: p_A
receives: p_B	$\longleftarrow$	computes: $p_B = g^{s_B}$

Step Two

computes: $K_A = p_B^{s_A}$	computes: $K_B = p_A^{s_B}$

Though we have shown the exchange of Alice and Bob's public keys in a sequential manner, this is not a requirement of the protocol. As Bob's transmission does not depend on Alice's, they can both send their messages at the same time.

The key computed by Alice is $K_A = p_B^{s_A} = g^{s_B s_A}$, while the key computed by Bob is $K_B = p_A^{s_B} = g^{s_A s_B}$. The two keys are therefore equal. The protocol has two steps (but only one round, as the number of rounds refers to the number of communication steps that are necessary), and each user needs to perform two exponentiations in $\mathbb{F}_p$, for coefficients in $\{2, \ldots, l-1\}$. We call s_A and s_B secret keys, and p_A and p_B public keys. Note that $s_A, s_B \in \{2, \ldots, l-1\}$, while $p_A, p_B \in G$. In the remainder of this thesis, we will refer to a finite field key pair of the form (s_i, p_i) with $p_i = g^{s_i}$ for a known public parameter g by the name Diffie-Hellman finite field key pair. We call K_A and K_B the common session key, as we have shown them to be equal.

As far as security goes, we now look at the information that is available to an eavesdropper Eve. It is assumed that she knows the setting parameters p, l, and g, as well as the exact nature of the protocol. The security of the Diffie-Hellman exchange protocol depends on the security of each of the two steps. In step one, each user computes a Diffie-Hellman finite field key pair (s_i, p_i), for $i \in \{A, B\}$. Eve can see the public keys p_A and p_B. The security of the protocol depends firstly on Eve's inability to compute the secret keys s_i from the public keys p_i, $i \in \{A, B\}$. Eve's capacity to find the secret keys is in fact measured by her capacity to break the discrete logarithm problem. Essentially, Eve is trying, given $p_i = g^{s_i}$ for known g and p_i, to find s_i, $i \in \{A, B\}$. This is the discrete logarithm problem and for a size of p of 1024 bits and a size of l of 160 bits, this step of the protocol is secure.

The second step presents a somewhat different security issue. Eve now possesses both public keys and is trying to compute the common conference key $K = K_A = K_B = g^{s_A s_B}$. This situation is in fact the setting of the computational Diffie-Hellman problem. Eve has $g, g^{s_A}, and g^{s_B}$ and wishes to calculate $g^{s_A s_B}$. As at the moment there is no other way to solve the CDHP than by means of the DLP, the security of the chosen parameters guarantees that the key exchange protocol presents a 80 bit security for Eve. In this case, therefore, just the computational Diffie-Hellman problem is relevant. However, should we want the protocol to be provably secure, we also need the indistinguishability property, and must also consider the DDHP.

On elliptic curves, the protocol is nearly identical. The public parameters of the setting are now q, l, P with q a prime or a prime power. We define E to be an elliptic curve over $\mathbb{F}_q$ and take the prime l to divide $\#E(\mathbb{F}_q)$. P is a point on E of order l and it generates a cyclic subgroup under point addition, which we call G. The protocol now proceeds as follows:

Alice		Bob
knows q, l, P, E		knows q, l, P, E

Step One

chooses $s_A \in \{2, \ldots, l-1\}$		chooses $s_B \in \{2, \ldots, l-1\}$
computes: $Q_A = s_A P$	$\longrightarrow$	receives: Q_A
receives: Q_B	$\longleftarrow$	computes: $Q_B = s_B P$

Step Two

computes: $K_A = s_A Q_B$	computes: $K_B = s_B Q_A$

Again, we have that $K_A = K_B$, as $K_A = s_A s_B P$ and $K_B = s_B s_A P$; we call $K = K_A = K_B$ the common session key. Identically to the protocol in finite fields, the Diffie-Hellman key exchange in elliptic curves has two steps, and each user performs two scalar multiplications on $E(\mathbb{F}_q)$, involving factors in $\{2, \ldots, l-1\}$. s_A and s_B are secret keys, while Q_A and Q_B are public keys. Note that, though s_A and s_B are integers as in the finite field case, the public keys are no longer integers modulo p, but points on the elliptic curve E. We refer to a pair of the form (s, Q) with $Q = sP$ by the name Diffie-Hellman elliptic curve key pair.

The security of the protocol is reduced, in a similar manner as presented above, to the ECDLP and the ECCDHP (as described in 4.3). We note that in both cases, the eavesdropper Eve has to work with points on the elliptic curve, i.e. she has to solve the problems: $P, Q_i \rightsquigarrow s_i$ and $P, Q_A, Q_B \rightsquigarrow K$. Since the security of the elliptic curve DLP and CDHP cannot be breached by the Index Calculus method, the parameters involved are much smaller, i.e. q and l are both 160 bits for security comparable to a finite field security of 1024 bits for p and 160 bits for l. Of course, the trade-off is that the individual group operations are more complicated on elliptic curves. We refer to the appendix for some computational details regarding the operations performed during the protocol.

In order to better show the correspondence between the two settings, we show in the following table the protocol as seen as only one of the parties, for example Alice. A clear parallel can be drawn between the finite fields and the elliptic curve approach now, based on the correspondence between integers modulo p and points on the elliptic curve E, as well as between multiplication/exponentiation in finite fields and point addition/scalar multiplication on elliptic curves.

Finite Fields	Elliptic Curves
knows p, l, g	knows q, l, P

Step One

chooses $s_A \in \{2, \ldots, l-1\}$ computes: $p_A = g^{s_A}$ receives: $p_B = g^{s_B}$	chooses $s_A \in \{2, \ldots, l-1\}$ computes: $Q_A = s_A P$ receives: $Q_B = s_B P$

Step Two

computes: $K_A = p_B^{s_A}$	computes: $K_A = s_A Q_B$

5.1.2 Tripartite Key Exchange

While the Diffie-Hellman key exchange is the best, most efficient, and most natural choice of protocol for the case where there are only two parties involved, tripartite key exchange is most naturally done with the aid of pairings. It has been mentioned before that the group operations in finite fields and on elliptic curves only take a single parameter and thus it is logical that the smallest key exchange unit comprises two users. Pairings, however, are bilinear and take two parameters; therefore the key exchange unit becomes three. The pairing equivalent of Diffie-Hellman is for three users and has been proposed by Joux ([19]).

The setting is as follows. We consider the parameters q, E, l, P of an elliptic curve setting, and denote by $\hat{e}$ a suitably modified pairing for a given embedding degree k such $\hat{e}(P, P) \neq 1$. The protocol now runs as follows:

Tripartite Key Exchange

- **Step** 1:

  Each user U_i, for $i = 1, 2, 3$ generates a Diffie-Hellman elliptic curve key pair (s_i, Q_i) and broadcasts Q_i. Take $U_0 = U_3$ and define all the indices modulo 3.

- **Step** 2:

  Each user U_i, with $i = 1, 2, 3$ computes the common session key:

$$K_i = e(Q_{i+1}, Q_{i+2})^{s_i} \quad . \tag{5.1}$$

This protocol is somewhat more complicated than the simple Diffie-Hellman key exchange, but it also contains two steps. By bilinearity, we have that $K_1 = \hat{e}(s_2 P, s_3 P)^{s_1} = \hat{e}(P, P)^{s_1 s_2 s_3}$, and the same reasoning applies to other two users. The common session key

is thus $K = K_1 = K_2 = K_3 = \hat{e}(P, P)^{s_1 s_2 s_3}$. Each user computes one scalar multiplication on elliptic curves, one pairing, and one exponentiation in $\mathbb{F}_{q^k}$.

The security of this key exchange protocol depends on the security of two problems in different settings. Firstly, the eavesdropper must not be able to compute the secret key s_i from Q_i for any $i \in \{1, 2, 3\}$. This is true as long as the parameters are taken to be large enough for the ECDLP (as we are dealing with Diffie-Hellman elliptic curve key pairs) to be unsolvable. Thus, both q and l can be of size 160 bits.

The second security issue comes in the computation of the key pair. This time, the eavesdropper has P, $Q_1 = s_1 P$, $Q_2 = s_2 P$, and $Q_3 = s_3 P$, and needs to calculate $\hat{e}(P, P)^{s_1 s_2 s_3}$. By the BCDHP, the key exchange protocol is secure as long as q^k has 1024 bits. Therefore, the size of q should be about $\frac{1024}{k}$ bits.

The tripartite key exchange protocol represents the smallest key exchange unit for pairings. It is in fact the pairing equivalent of the Diffie-Hellman key exchange. The following table shows the bipartite finite field key exchange in section 5.1.1 and the tripartite pairing key exchange, both described from the point of view of user 1 (equivalent to Alice in the notation of the previous section).

Finite Fields	Pairings
knows p, l, g	knows $q, l, P, E, \hat{e}, k$
Step One	
chooses $s_1 \in \{2, \ldots, l-1\}$ computes: $p_1 = g^{s_1}$ receives: $p_2 = g^{s_2}$	chooses $s_1 \in \{2, \ldots, l-1\}$ computes: $Q_1 = s_1 P$ receives: $Q_2 = s_2 P$ and $Q_3 = s_3 P$
Step Two	
computes: $K_1 = p_2^{s_1}$	computes: $K_1 = \hat{e}(Q_2, Q_3)^{s_1}$

The correspondence between the finite field environment and the pairing setting, therefore, is two to three users as a basic unit, and exponentiation to pairing computation. As the simple elliptic curve setting is nearly identical to the finite field setting, we do not include it in the table, mentioning that a similar parallel can be easily drawn between the two settings. Though the computation is harder in the case of pairings than for simple finite fields or elliptic curves, the setting parameters can be taken smaller, and also more users now have the possibility of sharing the same key than in the Diffie-Hellman bipartite key exchange.

An important consideration in the case of a protocol in the pairing setting is the existence of a distortion map. For the fast BN curves, no such maps exist, and therefore the protocol must be slightly altered to allow for this. We consider a pairing $e : G_1 \times G_2 \longrightarrow G_3$ with G_1 and G_2 both of prime order l and G_1 linearly independent from G_2 (we want to avoid having to use $e(P, P) = 1$). Linear independence means that the points $Q \in G_2$ cannot be written as rP for any positive integer r. Take P to be a generator of G_1 and S a generator of G_2. The

protocol changes as follows:

Modified Tripartite Key Exchange

- **Step** 1:

  User U_1 generates triplet (s_i, Q_i, R_i) with $Q_1 = s_1 P$ and $R_1 = s_1 S$. User U_2 generates (s_2, R_2) with $R_2 = s_2 S$. User U_3 generates (s_3, Q_3), with $Q_3 = s_3 P$. The values: Q_1, R_1, R_2, Q_3 are broadcast.

- **Step** 2:

  Each user U_i, with $i = 1, 2, 3$ computes the common session key:

$$K_1 = e(s_3 P, s_2 S)^{s_1} ,$$
$$K_2 = e(s_3 P, s_1 S)^{s_2} ,$$
$$K_3 = e(s_1 P, s_2 S)^{s_3} .$$

The common session key in this case is $K = e(P, S)^{s_1 s_2 s_3}$. The workload of the users is asymmetric, as one of the users must compute an additional scalar multiplication. We notice that the system parameters in this case must also include the second point S, though in this case the distortion map does not appear.

In what follows, we generalise the basic key exchange units presented in this section to the case of n users, with n arbitrarily large.

5.2 Multipartite Key Exchange in Various Settings

Like in the case of a tripartite key exchange, the main question regarding the development of an efficient protocol to generate a common conference key is how to arrange the participants and what setting would be most natural to use. Though finite fields provide often enough a good setting, the increasing demands in security make elliptic curves a more reliable choice for the future.

We now consider the case of n users wanting to communicate in a secure way. We note that for $n = 2$ and $n = 3$, the protocols of the previous sections are sufficient and a preferable choice. Let n therefore be an integer larger than 3. There are several arrangements to be considered, two of which are discussed in the following sections.

Finally, just as in the case of the tripartite key exchange protocol due to Joux, there are elliptic curves which do not support distortion maps, though they have computable pairings. For these elliptic curves, we must change the protocol. The necessary modifications are shown in the last section of this chapter.

5.2.1 BD I

The BD I key exchange protocol (named after its inventors Burmester and Desmedt – [7]) has the n participants arranged in a circle. Though the setting for the classical BD I is finite fields, Desmedt and Lange have extended it to pairings – see [9]. We therefore first consider the BD I protocol in finite fields.

The setting for the protocol is as considered in 3.1. We have the primes p and l such that $l|p-1$, and an element $g \in \mathbb{F}_p$ of order l generating a cyclic subgroup of $\mathbb{F}_p$, which we denote by G. The n users are denoted $U_1, U_2, \ldots, U_n$, and define $U_0 = U_n$. The indices are taken modulo n. The figure below shows this arrangement:

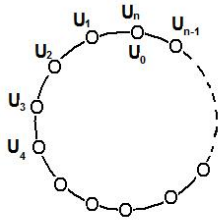

Figure 5.1: The BD I arrangement in finite fields

The protocol now works as follows:

Broadcast BD I in Finite Fields

- **Step** 1

  Each U_i computes a Diffie-Hellman finite fields key pair (r_i, z_i) and broadcasts z_i.

- **Step** 2

  Each U_i computes and broadcasts $X_i = \left(\frac{z_{i+1}}{z_{i-1}}\right)^{r_i}$.

- **Step** 3

  Each U_i computes the conference key:

$$K_i \equiv (z_{i-1})^{n\, r_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdot \cdots \cdot X_{i-2} \,. \tag{5.2}$$

We can show that all the users will compute the same key K, as given by the expression below:

$$K \equiv g^{r_1 r_2 + r_2 r_3 + \cdots + r_n r_1} \ . \tag{5.3}$$

Two different ways of proving this relation are given in [7] and [27] respectively. We will not repeat the proof here. We call this protocol fully contributory, as the secret exponent of each user appears in the expression of the common conference key. As opposed to the Diffie-Hellman finite fields key pair, the BD I protocol involves a key triplet (r_i, z_i, X_i), with r_i being secret and z_i and X_i public.

As it is visible, this protocol includes 3 steps, as opposed to only 2 in the bipartite and tripartite key exchange (and 2 communication rounds compared to 1). The quality of inter-party communication is therefore even more important in this case. If communications are known to be stable and fast, however, an even faster version of the protocol is available and described both in [7] and [27]. We show this turn-based form of the protocol in the next section. Without going too deep into the details at this point, we simply state that this different version trades individual computation expense for communication overhead, i.e. it contains more steps, but each participant needs to do less computations.

In this so-called broadcast version of the BD I protocol, each participant performs three exponentiations, one inversion, and $2n + 1$ multiplications. This boundary is reached if the implementation is done efficiently, using several computational tricks such as the Horner-like method for the computation of products of powers presented in the appendix. Depending on the programming platform, one could gain speed by trading the inversion for 3 multiplications, such as in Montgomery's trick (see the appendix).

The security of the BD I protocol in finite fields depends in steps 1 and 2 on the DLP, as in both cases a public parameter is taken to the secret exponent r_i. The CDHP offers security in the key computation, as the secrecy of the key lies in the $(z_{i-1})^{nr_i}$ factor, of which $(z_{i-1})^{r_i} = g^{r_{i-1} r_i}$ is unknown.

In finite fields therefore, the BD I protocol takes a circular arrangement regardless of the value of n (as long as $n > 2$). This reflects the linear structure of the key exchange unit in finite fields. In the pairing setting, however, the basic unit is the Joux triangle, and this is reflected in the user arrangement that Desmedt and Lange use in [9].

We consider now the setting with $n > 3$ participants and define $m = \lfloor \frac{n}{2} \rfloor$. When n is even, the first user is identified with the last user, as in finite field. Instead of linear communication, however, the Desmedt Lange (DL) arrangement features triangular cells, as shown in the figure below:

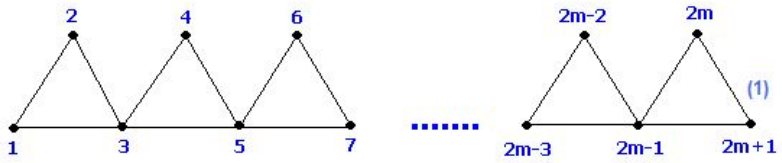

Figure 5.2: The BD I arrangement for parings

If the number of users n is odd, participant $n = 2m + 1$ exists and occupies the last vertex. In what follows we consider that the participants are denoted $U_1, \ldots U_n$, and vertex $U_{2m+1} = U_1$ if $n = 2m$ and $U_{2m+1} = U_n$ otherwise. The (publicly known) setting is now given as in section 3.3. Let q be a prime or prime power, and consider an elliptic curve E defined over $\mathbb{F}_q$. Furthermore, we have a prime $l|\#E(\mathbb{F}_q)$ and P a point on E of prime order l, generating a cyclic subgroup of l-torsion points. We are given an efficiently computable pairing $\hat{e}$ which includes in its definition the distortion map ϕ for a suitable embedding degree k.

The protocol now works as follows:

Broadcast BD I with Pairings

- **Step** 1

  Each user U_i, for $i = 1, 2, \ldots, n$ generates a Diffie-Hellman elliptic curve key pair (r_i, Z_i) and broadcasts Z_i.

- **Step** 2

  Each user U_i, with i odd and $i = 3, \ldots, 2m - 1$ computes and broadcasts:

$$X_i = \left(\frac{\hat{e}(Z_{i+1}, Z_{i+2})}{\hat{e}(Z_{i-2}, Z_{i-1})} \right)^{r_i} \tag{5.4}$$

  and X_i^{-1}.

- **Step** 3

  Each user U_i computes its conference key K_i as follows:

$$K_1 = (\hat{e}(Z_2, Z_3))^{(m-1)r_1} X_3^{m-2} X_5^{m-3} \ldots X_{2m-3},$$

$$K_i = T_i^{(m-1)r_i}(X_3 X_5^2 \dots X_{2j-1}^{j-1})^{-1}(X_{2j+1}^{m-j-1} X_{2j+3}^{m-j-2} \dots X_{2m-3}) \;\; i \neq 1. \qquad (5.5)$$

In the last equation, $j = \lfloor \frac{i}{2} \rfloor$ and $T_i = \hat{e}(Z_{i-1}, Z_{i+1})$ for i even and $T_i = \hat{e}(Z_{i-2}, Z_{i-1})$ for i odd.

It can be shown that all the users compute the same key, $K = \hat{e}(P, P)^d$, with $d = r_1 r_2 r_3 + r_3 r_4 r_5 + \dots r_{2m-3} r_{2m-2} r_{2m-1}$. This has been proven in [9]. As opposed to the BD I in finite fields, therefore, the DL protocol is not fully contributory, since the contributions of users U_{2m} and, if he exists, U_{2m+1} are not present in K. However, though not all the secret keys of the n users appear in the common conference key, the computations do depend on all these values. There are clearly two types of users as far as computation and storage are concerned.

The group which has a lighter computational burden is made up of users U_1, U_{2m+1}, and the even users. These users perform: one pairing, one scalar multiplication on elliptic curves, and $m - 2$ multiplications. They only use a Diffie-Hellman elliptic curve key pair. A group of users that have more computations to perform are the odd users with $U_3, \dots, U_{2m-1}$. We call this group the "odd" users, although users U_1 and U_{2m+1} are not included. These participants perform: 2 pairings, 2 inversions, 1 scalar multiplication, and 2 exponentiations, and they use a key triplet, (r_i, Z_i, X_i). As in the finite field case, implementing Montgomery's trick will exchange inversions for multiplications.

The security of the BD I protocol in the pairing setting depends in step 1 on the security of the ECDLP. In steps 2 and 3, the security is based on the assumption of the BCDHP for the pairing $\hat{e}$. Therefore the size of the parameter q^k should be at least 1024 bits.

The two protocols run in similar ways, with pairings replacing some of the finite field exponentiations required in the $\mathbb{F}_p$ setting. The linear key exchange unit is further replaced by Joux triangles. The main difference, however, is in the size of the parameters. While the main operation in the finite field environment is exponentiation in a 1024-bit finite field, in the pairing setting the size of the parameter q is only $\frac{1024}{k}$ bits. In the following tables, we show the correspondence between the contributions of an individual user in the classic finite field BD I and the contributions of an even and an odd user in the pairing setting.

To ease notation, we remark that usually all the indices are taken modulo n.

Finite Fields	Pairings – even user
knows p, l, g	knows $q, l, P, E, \hat{e}, k$

Step One

chooses $r_i \in \{2, \ldots, l-1\}$ computes: $z_i = g^{r_1}$ receives: z_{i-1}, z_{i+1}	chooses $r_i \in \{2, \ldots, l-1\}$ computes: $Z_i = r_i P$ receives: Z_{i-1}, Z_{i+1}

Step Two

computes: $X_i = \left(\frac{z_{i+1}}{z_{i-1}}\right)^{r_i}$ receives: $X_1, \ldots, X_n$	– receives: $\{X_3, X_3^{-1}, \ldots, X_{2m-1}, X_{2m-1}^{-1}\} \setminus \{X_i, X_i^{-1}\}$

Step Three

computes: $K_i = (z_{i-1})^{nr_i} X_i^{n-1} \ldots X_{i-2}$	computes: $K_i = \hat{e}(Z_{i-1}, Z_{i+1})^{(m-1)r_i}$ $(X_3 X_5 \ldots X_{i-1}^{\frac{i}{2}-1})^{-1} (X_{i+1}^{m-\frac{i}{2}-1} \ldots X_{2m-3})$

Similarly, for the odd participants we have:

Finite Fields	Pairings – odd user
knows p, l, g	knows $q, l, P, E, \hat{e}, k$

Step One

chooses $r_i \in \{2, \ldots, l-1\}$ computes: $z_i = g^{r_i}$ receives: z_{i-1}, z_{i+1}	chooses $r_i \in \{2, \ldots, l-1\}$ computes: $Z_i = r_i P$ receives: $Z_{i-2}, Z_{i-1}, Z_{i+1}, Z_{i+2}$

Step Two

computes: $X_i = \left(\frac{z_{i+1}}{z_{i-1}}\right)^{r_i}$ receives: $X_1, \ldots, X_n$	computes: $X_i = \left(\frac{\hat{e}(Z_{i+1}, Z_{i+2})}{\hat{e}(Z_{i-2}, Z_{i-1})}\right)^{r_i}$ and X_i^{-1}; receives: $\{X_3, X_3^{-1}, \ldots, X_{2m-1}, X_{2m-1}^{-1}\} \setminus \{X_i, X_i^{-1}\}$

Step Three

computes: $K_i = (z_{i-1})^{nr_i} X_i^{n-1} \ldots X_{i-2}$	computes: $K_i = \hat{e}(Z_{i-2}, Z_{i-1})^{(m-1)r_i}$ $(X_3 X_5 \ldots X_{i-2}^{\frac{i-1}{2}-1})^{-1} (X_i^{m-\frac{i-1}{2}-1} \ldots X_{2m-3})$

As it is visible, the two protocols are essentially the same, though their arrangements are slightly different. While the X_i in the classical finite fields is computed based on public keys

that are elements of $\mathbb{F}_p$, in the pairing setting, it is computed based on pairings. Similarly, the public keys z_i in finite fields are replaced by pairings in the computation of the key. Overall, the two settings present similar communication overhead, and comparative efficiency. In fact, [27] shows that, for very large values of n, the computation load is comparable even for a small value of the embedding degree k in the pairing setting. It is expected that for 128 bit security (equivalent to a field size of 12256 bits for finite fields), the implementation of the protocol in a $k = 12$ pairing environment on the fast BN curves would be much faster than in finite fields.

5.2.2 Turn-based BD I

As stated before, a turn-based version of the BD I protocol will ensure key exchange in more steps, but each of the n users involved will have less computations to perform. A version of the turn-based BD I protocol has already been presented in [7], and we repeat it here. This protocol takes place in a classical finite field setting as in section 3.1. The algorithm runs now as follows:

Turn-based BD I in Finite Fields

- **Step** 1

  Each U_i selects a random $r_i \in \{2, 3, \ldots, l-1\}$. Each user computes and multicasts $z_i = g^{r_i}$ to its two neighbours.

- **Step** 2

  Each U_i computes $X_i = \left(\frac{z_{i+1}}{z_{i-1}}\right)^{r_i}$.

- **Step** 3

  User U_1 initialises: $i \leftarrow 2$, $s_1 \leftarrow t_1 \leftarrow X_1$.

- **Step** $2 + i$

  User U_i receives: s_{i-1} and t_{i-1}, and computes and sends to U_{i+1}: $s_i = s_{i-1} \cdot X_i$ and $t_i = t_{i-1} \cdot s_{i-1}$. Do: $i \leftarrow i + 1$.

- **Step** $3 + n$

  User U_1 receives: s_n and t_n, and initialises: $j \leftarrow 2$ and $d_1 \leftarrow s_n t_n X_1^{-n}$.

- **Step** $2 + n + j$

  User U_j receives: s_n and d_{j-1}, and computes and sends to U_{j+1}: $d_j = s_n \cdot d_{j-1} \cdot X_j^{-n}$. Do: $j \leftarrow j + 1$.

- **Step** $2n + 3$

  Each user U_i, $i \in \{1, \ldots, n\}$ computes the key:

$$K_i = d_{i-1} \cdot (z_{i-1})^{n\, r_i} = (z_{i-1})^{n\, r_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdot \cdots \cdot X_{i-2} \,. \tag{5.6}$$

As stated within the protocol itself, there are $2n + 3$ steps in this protocol, though each of the computational rounds is simpler than its correspondent in the broadcast BD I algorithm from the previous section. In this case, the values of z_i need not be broadcast, but multicast to the neighbours of user U_i. Similarly, in step 2 the values of the X_i need not be broadcast, as they are only used by the user itself; however, we have left this computation as a separate step in order to preserve the resemblance with the broadcast protocol. After step two, each user computes: five multiplications, one inversion, and two exponentiations. As opposed to the broadcast version of BD I, the key computation is not left to every user individually, but each user contributes to the computation of all the keys. On the other hand, the key calculation remains secure, as the participants only use public parameters in the additional steps 3 and $2n + 2$.

We hereby present also the turn-based version of the BD I protocol in the pairing setting. For this, we consider the same setting as in section 3.3. The protocol has less steps than its equivalent in finite fields. Furthermore, an additional attempt has been made to make the computations asymmetric, so that the already heavier-loaded participants are responsible for most of the extra calculations. In this way, the users who have the lighter burden will have even less to do. In the pairing setting, the protocol works as follows:

Turn-based BD I with Pairings

- **Step** 1

  Each user U_i, for $i = 1, 2, \ldots, n$ generates a Diffie-Hellman elliptic curve key pair (r_i, Z_i) and multicasts Z_i to its neighbours.

- **Step** 2

  Each user U_i, with i odd and $i = 3, \ldots, 2m - 1$ computes:

$$X_i = \left(\frac{\hat{e}(Z_{i+1}, Z_{i+2})}{\hat{e}(Z_{i-2}, Z_{i-1})} \right)^{r_i} . \tag{5.7}$$

and X_i^{-1}.

- **Step** 3

  User U_3 initialises: $s_3 \leftarrow t_3 \leftarrow X_1$ and $j \leftarrow 5$.

- **Step** $2 + \frac{j-1}{2}$

  U_j receives s_{j-2} and t_{j-2}; then it computes and sends to U_{j+2}: $s_j = s_{j-2} \cdot X_j$ and $t_j = s_j \cdot t_{j-2}$. Do: $j \leftarrow j + 2$.

- **Step** m

  User U_{2m-3} receives s_{2m-5} and t_{2m-5}, then computes $s_{2m-3} = s_{2m-5}X_{2m-3}$ and $t_{2m-3} = s_{2m-3}t_{2m-5}$. User U_{2m-3} multicasts t_{2m-3} to U_1, U_2, and U_3.

- **Step** $m + 1$

  User U_3 initialises: $i = 5$, $k_4 = t_{2m-3}(X_3^{-1})^{m-1}$ and sends it to users U_5 and U_4.

- **Step** $m + \frac{i-1}{2}$

  User U_i receives k_{i-1}, then computes and sends to U_{i+1} and U_{i+2}: $k_{i+1} = k_{i-1} \cdot [(X_j)^{-1}]^{m-1}$. Do: $i \leftarrow i + 2$.

- **Step** $2m$

  Each user U_i with $i \in \{1, 2, \ldots, n\}$ computes the key:

$$
\begin{aligned}
K_1 &= t_{2m-3}\hat{e}(Z_2, Z_3)^{(m-1)r_1}; \\
K_i &= k_i(T_i)^{(m-1)r_i} \quad i \text{ even}; \\
K_i &= k_{i-1}(T_i)^{(m-1)r_i} \quad i \text{ odd}.
\end{aligned}
$$

  In the last equation, $T_i = \hat{e}(Z_{i-1}, Z_{i+1})$ for i even and $T_i = \hat{e}(Z_{i-2}, Z_{i-1})$ for i odd.

Once more, we point out that there are only $2m - 2$ communication rounds involved in this version of the protocol, and that only multicasts are necessary instead of broadcasts. The second public key X_i need also not be distributed to other users than user U_i itself; this step is once more set aside so as to show the resemblance with the broadcast version. In the turn-based protocol, the odd users compute four multiplications and two exponentiations

after step 2, while the even participants need to perform one multiplication and one exponentiation. The computed key is the same as in the broadcast setting. We can show this easily by noticing that $s_j = X_3 \dots X_j$ and $t_j = X_3^{\frac{j-1}{2}} \dots X_j$, for $j \in \{3, 5, \dots, 2m-3\}$. It also holds that $k_j = (X_3 \dots X_{j-1}^{\frac{j-2}{2}})^{-1}(X_{j+1}^{m-1-\frac{j}{2}} \dots X_{2m-3})$ for $j \in \{2, 4, \dots 2m\}$.

Therefore, in the key computation step, we have: $K_1 = X_3^{m-2} \dots X_j$, $K_i = T_i^{(m-1)r_i}$ $(X_3 \dots X_{i-1}^{\frac{i-2}{2}})^{-1}(X_{i+1}^{m-1-\frac{i}{2}} \dots X_{2m-3})$ for $i = 2j$ and $K_i = T_i^{(m-1)r_i}(X_3 \dots X_{i-2}^{\frac{i-3}{2}})^{-1}$. $(X_i^{m-1-\frac{i-1}{2}} \dots X_{2m-3})$ for $i = 2j+1$. This expression is equivalent to that of relation (4.5). The security is not compromised because the s_i's, t_i's, and k_i are all made up of public parameters.

The turn-based version of the BD I protocol – whether in finite fields or in the pairing setting – is a trade-off between the computational burden of each user and the number of interactions between the users. As opposed to the broadcast version, where the algorithm's speed depends on the speed of the participants' individual hardware, the turn-based version is only as fast as its slowest odd participant. However, it is no longer necessary to broadcast the values of the public keys generated in step 1; it suffices for each user to multicast its public key to its neighbour. It is paramount therefore that this system be implemented only in situations where it is known that all the users can interact efficiently and in a timely fashion. Therefore, using this version of the protocol is not recommended for wireless networks or mobile phones.

It is visible that in finite fields, the turn-based BD I protocol has more than twice the number of steps as the pairing-based version. In both versions, the values of the computed X_i need not be broadcast, as they are only used by the users who have computed them. Though there are a lot of computational steps involved, however, the number of steps per user is still quite small: three for each of the users in the finite fields setting, and three and one respectively for the heavily burdened and for the lightly burdened users in the pairing-based setting. We show a more detailed comparison in what follows. Once more, all the indexes are taken modulo n.

In order to make a comparison, we consider communication rounds rather than the steps as described above.

Finite Fields	Pairings – even user
knows p, l, g	knows $q, l, P, E, \hat{e}, k$

Round One

chooses $r_i \in \{2, \ldots, l-1\}$ computes: $z_i = g^{r_1}$ receives: z_{i-1}, z_{i+1}	chooses $r_i \in \{2, \ldots, l-1\}$ computes: $Z_i = r_i P$ receives: Z_{i-1}, Z_{i+1}

Round Two

computes: $X_i = \left(\frac{z_{i+1}}{z_{i-1}}\right)^{r_i}$	–
receives: s_{i-1}, t_{i-1}	–
computes: $s_i = s_{i-1} X_i$	–
$t_i = t_{i-1} s_i$	–
sends to U_{i+1}: s_i, t_i	receives: t_{2m-3}

Round Three

receives: d_{i-1}, s_n	–
computes: $d_i = s_n \cdot d_{i-1} \cdot X_i^{-n}$	–
sends to U_{i+1}: d_i	receives: k_i
computes: $K_i = d_{i-1} \cdot (z_{i-1})^{n\, r_i}$	computes: $K_i = k_i \hat{e}(Z_{i-1}, Z_{i+1})^{(m-1) r_i}$

Similarly, for the odd participants we have:

Finite Fields	Pairings – odd user
knows p, l, g	knows $q, l, P, E, \hat{e}, k$

Round One

chooses $r_i \in \{2, \ldots, l-1\}$ computes: $z_i = g^{r_1}$ receives: z_{i-1}, z_{i+1}	chooses $r_i \in \{2, \ldots, l-1\}$ computes: $Z_i = r_i P$ receives: Z_{i-1}, Z_{i+1}

Round Two

computes: $X_i = \left(\frac{z_{i+1}}{z_{i-1}}\right)^{r_i}$	computes: $X_i = \left(\frac{\hat{e}(Z_{i+1}, Z_{i+2})}{\hat{e}(Z_{i-2}, Z_{i-1})}\right)^{r_i}$ and X_i^{-1}
receives: s_{i-1}, t_{i-1}	receives: s_{i-2}, t_{i-2}
computes: $s_i = s_{i-1} X_i$	computes: $s_i = s_{i-2} X_i$
$t_i = t_{i-1} s_i$	$t_i = t_{i-2} s_i$
sends to U_{i+1}: s_i, t_i	sends to U_{i+2}: s_i, t_i
	receives: t_{2m-3}

Round Three

receives: d_{i-1}, s_n	receives k_{i-1}
computes: $d_i = s_n \cdot d_{i-1} \cdot X_i^{-n}$	computes: $k_{i+1} = k_{i-1} \cdot [(X_j)^{-1}]^{m-1}$
sends to U_{i+1}: d_i	sends to U_{i+2}: k_i
computes: $K_i = d_{i-1} \cdot (z_{i-1})^{n\, r_i}$	computes: $K_i = k_{i-1}(T_i)^{(m-1)r_i}$

5.2.3 BD II

Another possible arrangement for the general key exchange for n users is a tree arrangement. Classically, the setting for the BD II is finite fields, based on the linear key exchange unit that we saw in the BD I as well.

We consider the same p, l, g setting as in the previous section. The n users, $U_1, \ldots, U_n$ with $n > 2$ are arranged in a binary tree whose root has been removed and the top two users have been connected. Each user has a level $\text{Lvl}_{U_i} = \lfloor \log_2(i+1) \rfloor$. Each user has a parent – for every user U_i with $i \notin \{1, 2\}$, we define its parent to be $U_{\lfloor \frac{i-1}{2} \rfloor}$, and we set U_1 and U_2 to be each other's parent – and apart from the users in the last level – also called leaves – each user has one or two children, which are U_{2i+1} (left child) and U_{2i+2} (right child). We denote the list of ancestors of a user U_i all its ancestors, including the user himself, but excluding both users 1 and 2: $\text{Anc}_{U_i} = \{U_i, U_{\lfloor \frac{i-1}{2} \rfloor}, \ldots, U_{\lfloor (i-1)/(2^{\text{lvl}_{U_i}}-1) \rfloor}\} \setminus \{U_1, U_2\}$. The arrangement is shown in the figure below.

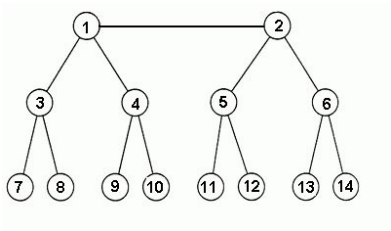

Figure 5.3: The BD II in finite fields

For this figure, the participants in level 3 are the leaves. User U_5 has as parent U_2 and its children are: $\text{lchild}(U_5) = U_{11}$ and $\text{rchild}(U_5) = U_{12}$. The set of ancestors of user U_{11} is: $\text{Anc}_{U_{11}} = \{U_{11}, U_5\}$. We note that this structure, though it might seem more artificial than the circular arrangement in BD I, is actually inherent in certain types of communications, for example in hierarchic structures.

Once this arrangement and the public setting parameters are known to all the users, the protocol works as follows:

The BD II protocol

- **Step** 1

  Each U_i computes a Diffie-Hellman finite fields key pair (r_i, z_i) and sends it to his parent and children.

- **Step** 2

  Each U_i apart from the leaves computes and multicasts to its children $X_{\text{lchild}(U_i)} = \left(\frac{z_{\text{parent}(U_i)}}{z_{\text{lchild}(U_i)}}\right)^{r_i}$ and $X_{\text{rchild}(U_i)} = \left(\frac{z_{\text{parent}(U_i)}}{z_{\text{rchild}(U_i)}}\right)^{r_i}$.

- **Step** 3

  Each U_i computes the conference key:

$$K_i = (z_{\text{parent}(U_i)})^{r_i} \cdot \prod_{j \in \text{Anc}_{U_i}} X_j \ . \tag{5.8}$$

In this protocol, each participant uses a key triplet (r_i, z_i, X_i); however, as opposed to the previous protocols, the second public key X_i is not computed by the user itself, but by its parent. Therefore, users U_1 and U_2 compute the keys X_3, X_4 and X_5, X_6 respectively, but not their own X_1 and X_2. The protocol is not fully contributory, as the common key computed by all the users is $K = g^{r_1 r_2}$ (as it was proven in for instance [10]). However, as noted also in the case of the BD I protocol in the pairing setting, the computation of the key requires contributions from all the users.

The security of the protocol is also proven formally in [10]. We mention here only that the recommended sizes of the system parameters for an 80–bit security are those given in section 3.4. The computational burden of each user depends on its level. The leaves, which do not have to compute their own X_i, perform two exponentiations and $\text{lvl}_{U_i} - 1$ multiplications. Each user needs only input from its direct ancestors; the cardinality of Anc_{U_i} is $\lfloor \log_2(i+1) \rfloor$. Therefore the complexity of the protocol is now logarithmic, instead of linear in n. All the users that are not leaves compute an extra inversion, multiplication, and exponentiation per each child.

The BD II protocol can be translated to the pairing setting, an idea which has been presented by Desmedt and Lange in [9]. In the finite field setting, the key exchange unit remains

somewhat linear, i.e. each participant is connected to a single parent and two children. In the pairing setting, the structure becomes more convoluted.

We consider a situation when $n > 3$ users $U_1, \ldots, U_n$ wish to communicate in a secure fashion, in a setting as described in section 3.3. Instead of two roots, as in the finite field environment, the pairing case includes three special users U_1, U_2, U_3. Each user has four children, two left children – U_{4i+2} and U_{4i+3} – and two right children – U_{4i} and U_{4i+1}. The parent of all the users apart from the three users is given by $U_{\lfloor i/4 \rfloor}$. For a user U_i, the other child of parent(U_i) from the same branch is called U_i's sibling, and it is denoted sibling(U_i). The set of ancestors of a user U_i is defined as before: the user itself and all its ancestors, but not including users U_1, U_2, and U_3. This arrangement may be clearer in the figure below.

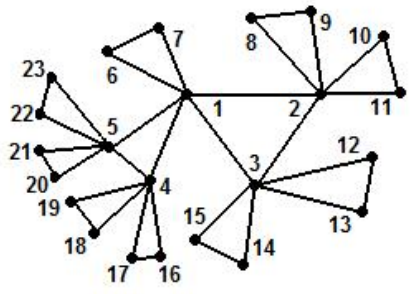

Figure 5.4: The BD II arrangement for parings

For the users in the first level, it holds: parent(U_1) $= U_2$, parent(U_2) $= U_3$, and parent(U_3) $= U_1$, while sibling(U_1) $= U_3$, sibling(U_2) $= U_1$, and sibling(U_3) $= U_2$.

As an example, we consider user U_4. For this user, parent(U_4) $= U_1$, sibling(U_4) $= U_5$, the left children are: lchild1(U_4) $= U_{18}$ and lchild2(U_4) $= U_{19}$, while the right children are: rchild1(U_4) $= U_{16}$ and rchild2(U_4) $= U_{17}$. The set of ancestors of, for instance, user U_{17} is: $\text{Anc}_{U_{17}} = \{U_{17}, U_4\}$. In their paper [10], Desmedt and Lange have suggested the idea of the protocol for a set of participants who all have two children each. This document tries to generalise their approach to any number of users.

We consider the general pairing setting described in 3.2. The protocol now proceeds as follows:

The BD II protocol in a pairing setting

- **Step** 1

  Each U_i computes a Diffie-Hellman elliptic curve key pair (r_i, Z_i) and sends it to his parent, sibling, and children.

- **Step** 2

  Each U_i apart from the leaves computes and multicasts to its descendants $X_{\text{lchildren}(U_i)} = \left(\frac{\hat{e}(Z_{\text{parent}(U_i)}, Z_{\text{sibling}(U_i)})}{\hat{e}(Z_{\text{lchild1}(U_i)}, Z_{\text{lchild2}(U_i)})}\right)^{r_i}$ and $X_{\text{rchildren}(U_i)} = \left(\frac{\hat{e}(Z_{\text{parent}(U_i)}, Z_{\text{sibling}(U_i)})}{\hat{e}(Z_{\text{rchild1}(U_i)}, Z_{\text{rchild2}(U_i)})}\right)^{r_i}$. If the user has only one child, it uses that child's public key twice in the denominator. For example, let us assume that the user only has a single right child, namely $\text{rchild1}(U_1)$. Then $X_{\text{rchild1}(U_i)} = \left(\frac{\hat{e}(Z_{\text{parent}(U_i)}, Z_{\text{sibling}(U_i)})}{\hat{e}(Z_{\text{rchild1}(U_i)}, Z_{\text{rchild1}(U_i)})}\right)^{r_i}$.

- **Step** 3

  Each U_i computes the conference key:

$$K_i = (\hat{e}(Z_{\text{parent}(U_i)}, Z_{\text{sibling}(U_i)}))^{r_i} \cdot \prod_{j \in \text{Anc}_{U_i}} X_j \ . \tag{5.9}$$

  In the key computation, if the user has no sibling, then the user uses his own public key, instead computing:

$$K_i = (\hat{e}(Z_{\text{parent}(U_i)}, Z_{U_i}))^{r_i} \cdot \prod_{j \in \text{Anc}_{U_i}} X_j \ . \tag{5.10}$$

Just as in the case of finite fields, the third key in the key triplet (r_i, Z_i, X_i) is computed by the parent of U_i and not U_i itself. The common conference key computed by all the participants is $\hat{e}(P, P)^{r_1 r_2 r_3}$. The protocol therefore is once more not fully contributory, though the contributions of all the ancestors are used in the computation of the key. The security of the protocol can be proven similarly as for the finite field case; in a pairing setting, however, the size of the parameters is much smaller.

The leaves each have to compute: 1 scalar multiplication on the elliptic curve, 1 pairing, 1 exponentiation, and $2\log_4(n) - 1$ multiplications. By contrast, regular users with $3 - 4$ children need to compute 2 additional pairings, 2 additional inversions, 2 multiplications, and 2 exponentiations. Each user depends on the contributions of its ancestors, thus the complexity of the protocol is logarithmic, $O(\log_4(n))$.

The two protocols can be even more effectively compared if we show the contributions of an individually chosen, non-leaf and non-root user U_i:

Finite Fields	Pairings
knows p, l, g	knows $q, l, P, E, \hat{e}, k$

Step One

chooses $r_i \in \{2, \ldots, l-1\}$ computes: $z_i = g^{r_1}$ receives: $z_{\text{parent}(U_i)}, z_{\text{rchild}(U_i)}, z_{\text{lchild}(U_i)}$	chooses $r_i \in \{2, \ldots, l-1\}$ computes: $Z_i = r_i P$ receives: $Z_{\text{parent}(U_i)}, Z_{\text{rchild1}(U_i)}, Z_{\text{rchild2}(U_i)}$ $Z_{\text{sibling}(U_i)}, Z_{\text{lchild1}(U_i)}, Z_{\text{lchild2}(U_i)}$

Step Two

computes: $X_{\text{lchild}(U_i)} = \left(\frac{z_{\text{parent}(U_i)}}{z_{\text{lchild}(U_i)}}\right)^{r_i}$ computes: $X_{\text{rchild}(U_i)} = \left(\frac{z_{\text{parent}(U_i)}}{z_{\text{rchild}(U_i)}}\right)^{r_i}$ receives: X_j for $j \in \text{Anc}_{U_i}$	computes: $X_{\text{lchildren}} = \left(\frac{\hat{e}(Z_{\text{parent}(U_i)}, Z_{\text{sibling}(U_i)})}{\hat{e}(Z_{\text{lchild1}(U_i)}, Z_{\text{lchild2}(U_i)})}\right)^{r_i}$ computes: $X_{\text{rchildren}} = \left(\frac{\hat{e}(Z_{\text{parent}(U_i)}, Z_{\text{sibling}(U_i)})}{\hat{e}(Z_{\text{rchild1}(U_i)}, Z_{\text{rchild2}(U_i)})}\right)^{r_i}$ receives: X_j for $j \in \text{Anc}_{U_i}$

Step Three

computes: $K_i = (z_{\text{parent}(U_i)})^{r_i} \prod_{j \in \text{Anc}_{U_i}} X_j \mod p$	computes: $K_i = \hat{e}(Z_{\text{parent}(U_i)}, Z_{\text{sibling}(U_i)})^{r_i} \prod_{j \in \text{Anc}_{U_i}} X_j$

Once more, the two protocols are very similar, though the computations are done in different settings. As in the case of the BD I, the pairing version has as few computations of pairings as possible. In both protocols, the third set of keys X_i is computed by the parents of the users, and not by the users themselves. In the case of the BD I protocol, the complexity is linear in the number of users; for the BD II protocol, the complexity is $O(\log_2(n))$ in finite fields and $O(\log_4(n))$ in the pairing setting. Furthermore, most of the users (about half in the finite fields case and about $\frac{3}{4}n$ in the pairing setting) are leaves and thus do not have to compute their own X_i. If one chooses the fast, BN curves setting, it is very possible for the protocol to run faster with pairings than with finite fields. In fact, the larger the value of n, the larger the difference in computation speed.

5.2.4 Modified Key Exchange

In the case of BN curves and other curves on which distortion maps cannot be defined, we consider as additional setting parameter the generator of the group G_2 – the group where the second argument of the pairing is taken from. We denote this generator by S. The modifications required by the BD I protocol will be the same for both the broadcast and the turn-based version, as they only concern the key generation step and sometimes the order of the arguments in the computation of the pairing.

We recall that the BD I arrangement for pairings is based on triangular units, as in the figure below. The easiest way to modify the protocol would be to have all users compute key

triplets (s_i, Z_i, R_i) with $Z_i = s_i P$ and $R_i = s_i S$; however, there exist also more efficient ways to achieve this.

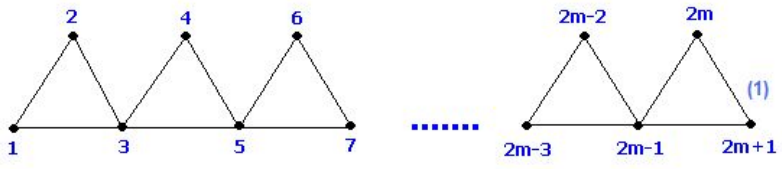

Figure 5.5: The BD I arrangement for parings

In what follows, we present the modified broadcast BD I protocol and state that the same modifications will achieve the successful modification of the turn-based version. We use the notations: Z_i for P-based pubic keys, and R_i for S-based public keys. Let $m = \lfloor \frac{n}{2} \rfloor$ and if n is even, take $U_{2m+1} = U_1$.

Modified Broadcast BD I with Pairings

- **Step** 1

  Each user U_i, for $i \in \{1, 3, \ldots, 2m+1\}$ and $i \bmod 4 = 1$ generates a triplet (s_i, Z_i, R_i). If U_{2m+1} does not exist, it is identified with user U_1 after key generation, and so has the first user's keys. Each even user U_i with $i = 2, 4, \ldots, 2m$ generates the key pair (s_i, Z_i). Each user U_i with $i \in \{1, 3, \ldots, 2m+1\}$ and $i \bmod 4 = 3$ generates the key pair (s_i, R_i).

- **Step** 2

  Each user U_i, with i odd and $i = 3, \ldots, 2m-1$ computes and broadcasts:

  $$X_i = \left(\frac{\hat{e}(Z_{i+1}, R_{i+2})}{\hat{e}(Z_{i-1}, R_{i-2})} \right)^{s_i} \tag{5.11}$$

  and X_i^{-1}.

- **Step** 3

  Each user U_i computes its conference key K_i as follows:

  $$K_1 = (\hat{e}(Z_2, R_3))^{(m-1)s_1} X_3^{m-2} X_5^{m-3} \ldots X_{2m-3},$$

$$K_i = T_i^{(m-1)s_i}(X_3X_5^2 \dots X_{2j-1}^{j-1})^{-1}(X_{2j+1}^{m-j-1}X_{2j+3}^{m-j-2} \dots X_{2m-3}) \quad i \neq 1. \tag{5.12}$$

In the last equation, $j = \lfloor \frac{i}{2} \rfloor$ and $T_i = \hat{e}(Z_{i-1}, R_{i+1})$ for i even and $T_i = \hat{e}(Z_{i-1}, R_{i-2})$ for i odd.

The resulting common conference key is $K = e(P, S)^d$ with $d = s_1s_2s_3 + s_3s_4s_5 + ... + s_{2m-3}s_{2m-2}s_{2m-1}$. The users that compute an extra scalar multiplication are users U_i with i odd and $i \bmod 4 = 1$; their proportion is maximal when the number of users $n \equiv 1 \bmod 4$, and then it is equal to $\frac{n-1}{4} + 1$. After the key generation step, the general rule is for the first argument of each pairing to contain P and the second argument to contain S. For this purpose, the arguments of the pairings in the denominator of X_i are switched. The same strategy can be applied in the turn-based protocol.

For the BD II protocol, the situation is slightly more complicated. We remind the reader of the general arrangement of the users. Instead of two roots, as in the finite field environment, we consider three special users U_1, U_2, U_3. The left children of user U_i are U_{4i+2} and U_{4i+3}; its right children are: U_{4i} and U_{4i+1}. The parent of all the users apart from the three special users is given by $U_{\lfloor i/4 \rfloor}$. U_i's sibling is the other child of parent(U_i) from the same branch, and it is denoted sibling(U_i). The set of ancestors of a user U_i contains U_i and all its ancestors, but not including users U_1, U_2, and U_3.

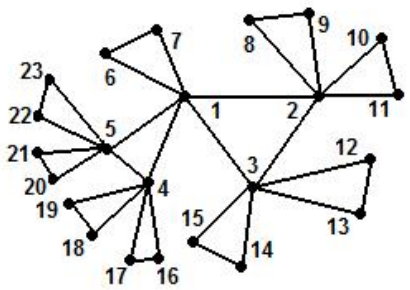

Figure 5.6: The BD II arrangement for parings

Let the cardinality of the set of ancestors of U_n be m. Let the set $I = \{i_1, ..., i_m, i_{m+1}\}$ include the indices of all these users in decreasing order of their indices and i_{m+1} is index of the parent of U_{i_m}, one of the users U_1, U_2, U_3. So $i_1 = i$, i_2 is the index of the parent of U_i, and so forth. The notation of the private and public key pairs remains as above. Furthermore, if U_i is a user, we denote par_i the index of its parent. So U_{par_i} is the parent of U_i. In this setting, the protocol is modified as follows:

The BD II protocol in a pairing setting

- **Step** 1

  For each user U_i, we find $j \in I$ such that $i_{j+1} < i \leq i_j$. If j is odd, all U_i with $i \bmod 2 = i_{j+1} \bmod 2$ that have siblings generate key pair (s_i, R_i) and all U_i with $i \bmod 2 \neq i_{j+1} \bmod 2$ that has siblings generate key pair (s_i, Z_i). If j is even, all U_i generate key triplets (s_i, Z_i, R_i). If $i_{m+1} = 3$ and m is even, U_1 generates a key triplet. All users U_i without siblings generate key triplets. Each user sends its public keys to its parent, siblings, and children.

- **Step** 2

  Each U_i apart from the leaves computes and multicasts to its descendants $X_{\text{lchildren}(U_i)} = \left(\frac{\hat{e}(Z_{\text{parent}(U_i)}, R_{\text{sibling}(U_i)})}{\hat{e}(Z_{\text{lchild1}(U_i)}, R_{\text{lchild2}(U_i)})}\right)^{s_i}$ and $X_{\text{rchildren}(U_i)} = \left(\frac{\hat{e}(Z_{\text{parent}(U_i)}, R_{\text{sibling}(U_i)})}{\hat{e}(Z_{\text{rchild1}(U_i)}, R_{\text{rchild2}(U_i)})}\right)^{s_i}$. If the user has only one child, it uses that child's public key twice in the denominator. For example, let us assume that the user only has a single right child, namely $\text{rchild1}(U_1)$. Then $X_{\text{rchild1}(U_i)} = \left(\frac{\hat{e}(Z_{\text{parent}(U_i)}, R_{\text{sibling}(U_i)})}{\hat{e}(Z_{\text{rchild1}(U_i)}, R_{\text{rchild1}(U_i)})}\right)^{s_i}$. The arguments of the pairings can be inverted so as to use the keys that the neighbouring users have.

- **Step** 3

  Each U_i computes the conference key:

$$K_i = (\hat{e}(Z_{\text{parent}(U_i)}, R_{\text{sibling}(U_i)}))^{s_i} \cdot \prod_{j \in \text{Anc}_{U_i}} X_j \ . \tag{5.13}$$

  In the key computation, if the user has no sibling, then the user uses his own public key, instead computing:

$$K_i = (\hat{e}(Z_{\text{parent}(U_i)}, R_{U_i}))^{s_i} \cdot \prod_{j \in \text{Anc}_{U_i}} X_j \ . \tag{5.14}$$

The common conference key computed by all the participants is $\hat{e}(P, S)^{r_1 r_2 r_3}$. We point out that most of the users will compute only a key pair, and only a few users will compute an

extra scalar multiplication. Every other layer of users will compute a key triplet, beginning with the second layer. Two siblings will either have different types of public keys (one being P-based and one being S-based) or will both have both types of public keys.

Chapter 6

Authentication and the Katz Yung Compiler

The protocols already described so far are key exchange (KE) protocols. It is useful to mention here what this entails and how the protocol can be made more secure by means of *authentication*. For years, authentication has been a topic of great interest, as it offers a way of proving one's identity within any communications' environment.

We define first the notions of *active* or *passive* adversaries. In public key cryptography we assume the presence of an adversary Eve, who has access to the communication between the parties involved. If the adversary is modelled so that she can only eavesdrop (i.e. without taking part in the communication), she is called a *passive adversary*. If Eve is assumed to be able to intervene in the communication between the parties involved (for instance by trying to impersonate one of the participants in the communication and sending messages on their behalf) then the adversary is called *active*.

A regular key exchange (KE) protocol should be generally safe against passive attackers. For example, a simple Diffie-Hellman exchange is safe against passive attacks, as solving the DLP, CDHP, and DDHP problems is computationally infeasible. When we have discussed the security of the previously presented protocols, we have shown them to be secure against passive attacks.

However, an active attacker can simply intercept the messages sent by, say, party A, and send a message of her choice instead. For example, let us assume that we have a simple protocol, where parties A and B choose secret keys and compute the corresponding public keys, which they send to each other. However, Carol could intercept the message from party A and instead choose a secret key of her own. It can then send party B the corresponding public key and thus only Carol and party B can perform encryption and decryption on the future messages. Similarly, by sending her own public key to party A, Carol will now be able to intercept and decrypt every message sent from one party to the other and substitute it with a message that she chooses and encrypts. Parties A and B will both remain unaware of Carol's existence in this case, as the communication is maintained. This is called a man-in-the-middle attack.

On the other hand, an authenticated key exchange (AKE) protocol has to be secure also against active attacks. These protocols require some means of authentication, meaning that the different parties can prove they are who they claim to be. An important result was

published by Katz and Yung in [21] – a compiler that turns any KE protocol into an AKE protocol. In what follows, we describe the general idea of this compiler.

We assume that there are n parties $U_1, \ldots, U_n$ trying to communicate securely over an insecure connection. We denote the set of users by $\mathcal{U}$. These parties are all assumed to be present in the same communication session. Let us assume there exists a KE protocol $\mathcal{P}$ for the key exchange between these parties. Then the Katz-Yung compiler will turn this protocol into an AKE protocol $\mathcal{P}'$. The j^{th} message sent by user U in this arrangement is indexed as $U|j|m$, where U denotes the user j, and m is the message itself. We define a signing function based on a secret key, SK_U and a verifying function based on a public key PK_U.

Binding public keys to identities can present some difficulties. Generally speaking, we simply assume that all the users have all the public keys of all their peers. In order to bind public keys to identities, one uses a Public Key Infrastructure (PKI). The user must generate a secret and a public key of its own, and then be granted a certificate from a trusted Certificate Authority; this signature proves that the identity truly corresponds to the public key. Though the process of running proper PKI can be too cumbersome to be included in any communication over the internet, it should certainly be involved in larger conferences that require a higher level of security.

Authentication requires signing messages. There are three algorithms involved in digital signatures (see [24]): key generation algorithms, signing algorithms, and verification algorithms. The key-pair consists of a private key S and a public key P. A message m can be signed by means of S and the signing algorithm (denoted by Sign); the output is a signature σ. Then the verification algorithm (denoted by Verify) uses P and m to check that σ is the signature generated for m with the secret key.

We now return to the Katz Yung compiler. During the initialisation of the protocol $\mathcal{P}$, each user U generates an extra secret key SK'_U (apart from any secret key generation required by the protocol $\mathcal{P}$). This user calculates then the public key PK'_U. These keys are used for signing and verifying subsequent communication. The compiler described below assumes the existence of this extra pair of secret and public keys.

The compiler that performs the transformation between $\mathcal{P}$ and $\mathcal{P}'$ in the following way:

Katz-Yung Compiler

1. Each user $U_i \in \mathcal{U}$ generates a random $r_i \in \{0,1\}^k$ and broadcasts the message: $U_i|0|r_i$. The message is therefore given order number 0 – an initiation value. All partners involved in this session store the values of the partners and their random values in a variable: $\text{nonces}_U = U_1|r_1|U_2|r_2|\ldots|U_n|r_n$.

2. The protocol $\mathcal{P}$ is executed, altered as follows:

   - Whenever user U wants to broadcast message $U|j|m$ by means of the protocol $\mathcal{P}$, it computes first $\sigma = \text{Sign}_{\text{SK'}_U}(j|m|\text{nonces}_U)$, where $\text{Sign}_{\text{SK'}_U}$ is the signing

function executed under the secret key $\mathrm{SK'}_U$. The user broadcasts $U|j|m|\sigma$.

- If a message $V|j|m|\sigma$ is received, the following actions will be performed: (1) : checking that $V \in \mathcal{U}$; (2) : it is checked that there has already been a message number $j-1$ sent by user V, but no message with number j yet; (3) : the signature is verified, i.e. it is checked that $\mathrm{Verify}_{\mathrm{PK'}_U}(j|m|\mathrm{nonces}_U, \sigma) = 1$. Herein, $\mathrm{Verify}_{\mathrm{PK'}_U}(a, s)$ is the verification procedure that checks the validity of s as a signature on a. It should hold that $s = \mathrm{Sign}_{\mathrm{SK'}_U}(a)$. If any of these 3 verification steps is not completed, then the protocol resets. Otherwise, the protocol continues as before.

3. If the protocol has not been reset yet, then each user may thereafter compute the shared secret key as instructed by $\mathcal{P}$.

Katz and Yung prove that if the KE protocol $\mathcal{P}$ is secure against passive adversaries, then the AKE protocol $\mathcal{P}'$ is also secure against active adversaries. The Katz-Yung compiler can be applied to the any of the protocols presented before, in order to obtain security against active adversaries. However, the complexity of the Katz-Yung compiler is $O(n)$, therefore a KE protocol with smaller complexity will not benefit from this means to turn it into an AKE protocol.

Chapter 7

Multi-partite Key Exchange in JCrypTool

An important part of the master project associated with this thesis was the implementation of the described cryptographic protocols in JCrypTool. This Java development branch of the CrypTool initiative already contains numerous algorithms, from the classical Caesar cipher, Viginere encryption, or a one-time pad, to AES, RSA, and zero-knowledge protocols. Apart from the user interface and user manual, JCrypTool also describes mathematical background for each of its implemented components and methods.

The current version of JCrypTool is milestone2, which has been released at the end of August 2008. This version of the project can be downloaded on Source Forge [18]. This version does not yet include the key exchange component, which is still at its beginning.

This chapter will be structured as follows: Initially a brief description of the author's contribution to the project is given in 7.1, then an interface is shown and described in more detail in 7.2, and finally a few comments are reserved for future development in 7.3.

7.1 Personal Contributions to JCrypTool

The JCrypTool project is based on plug-ins and features including plug-ins. The final output of the GUI's provided for key exchange are therefore feature projects in Java, and they include view-based plug-ins. The development of these plug-ins contains two aspects: the interface aspect and the protocol aspect. The following protocols were targeted for implementation and integration into JCrypTool at the beginning of this project:

- Authenticated pairing based tripartite key exchange (Joux' protocol).
- Authenticated broadcast BD I in finite fields and pairings.
- Authenticated broadcast BD II in finite fields and pairings.

It was later agreed upon that the authenticated Diffie-Hellman key exchange protocol should also be implemented, both on finite fields and on elliptic curves. For the pairing

based protocols, we decided to use the Tate pairing (see section 3.3) on supersingular curves with embedding degree 2.

In order to develop these protocols, the following structures were implemented, in order to complement the tools already provided by Java and the external FlexiProvider toolkit [15]:

- A parameter generator for finite fields, which takes as input the key size (size of l), and outputs a parameters object including values for p, l, and g as required by the finite field setting (see section 3.1).
- A parameter generator for elliptic curves with pairings (i.e. curves with small embedding degree). This generator takes as input the size of q and outputs a parameters object including values for q, l, and the coordinates of P on an elliptic curve E that is preset in the generator. More details regarding this setting can be found in sections 3.2 and 3.3.
- Elliptic curve arithmetic for an elliptic curve over $\mathbb{F}_p$ for p a large prime. Point addition and point doubling are done with Jacobian coordinates, but the possibility is given to return the point also in affine coordinates.
- Basic polynomial arithmetic for polynomials with coefficients either in $\mathbb{Z}$ or in $\mathbb{F}_p$. Reduction modulo another polynomial f is provided, together with multiplication, exponentiation, and inversion modulo f.
- A very basic construction of points on $E(\mathbb{F}_{p^2})$ was implemented as default. This structure can be extended to $E(\mathbb{F}_{p^n})$.
- A Tate pairing implementation is given for embedding degree 2 (with distortion maps). This implementation is based on Miller's algorithm. A Weil pairing implementation is amongst the first items on the list of future developments (see section 7.3).
- Implementations of the bipartite (Diffie-Hellman), Joux, Broadcast BD I, and BD II protocols as described above.
- Authentication for an arbitrary message m with default method SHA-1 followed by RSA.

We implemented the user interfaces for the following protocols:

- The Diffie-Hellman key exchange interface in finite fields. Available interaction: key size, authentication (impersonation attacks), manual/automatically-generated input.

- The Broadcast BD I in finite fields. Available interaction: parameter size (two choices, 5-bit parameters and 1024 bit parameters), number of users $n > 2$.
- The Broadcast BD I in the pairing setting, embedding degree 2 with default curve. Available interaction: parameter size (two choices, 8-bit parameters and 512-bit parameters), number of users $n > 3$.

7.2 Sample Interface

As there are many aspects of key exchange that can be interesting to see, each of the protocol interfaces attempts to show new facets of key exchange, and different parameters are varied. We show here only one of these interfaces and mention that all the interfaces will all be made as user-friendly as possible, so that a greater understanding of key exchange might follow as naturally as possible.

As key exchange is not yet included in the JCrypTool product, its location might be subject to change. The user can choose between protocols from a menu; what we describe in the following pages, however, is a brief overview of the interface that becomes accessible once the BD I protocol with pairings has been chosen.

There are two types of parameters that can be set by the user: the number of users and the setting parameters. The elliptic curve E is set by default to have the short Weierstrass equation $y^2 = x^3 + x$, and the embedding degree $k = 2$. There are two choices of parameter sizes: high and low. The values of q, l, and the coordinates of P are automatically chosen for the specified parameter size. Low security equates: $q = 103$, $l = 13$, and $P = (26, 35)$. For a high security level, the parameters are automatically set to those mentioned in [27].

Once the setting parameters are set, the user can then choose a number of users $n > 3$. Special values of n are 4, 5, and 6; for these particular parameters, special protocol illustrations and explanations follow. Generalised illustrations are also drawn for any value of n. Figure 7.1 shows an example of the illustration for $n = 5$.

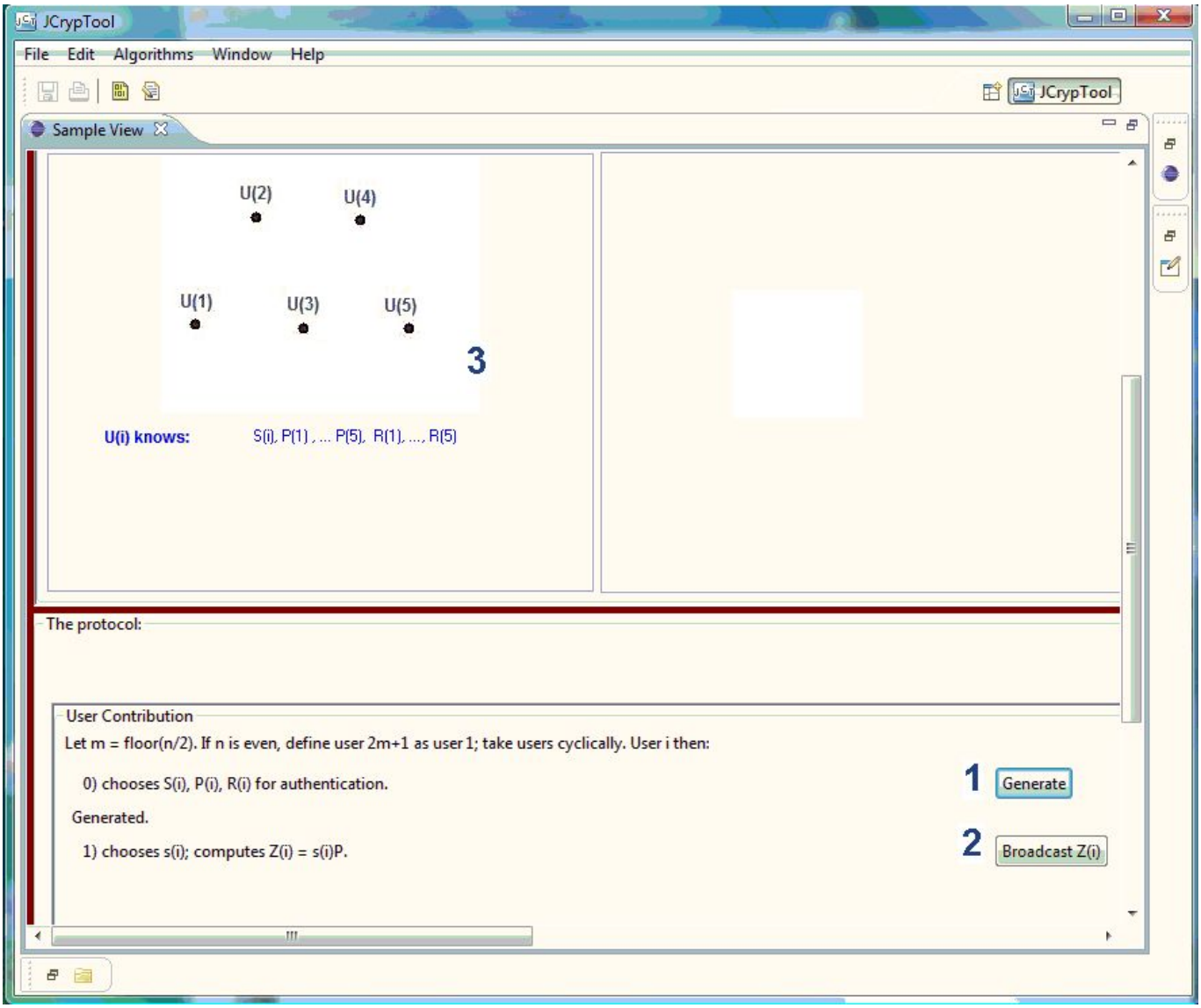

Figure 7.1: The Case $n = 5$

The illustration panel, denoted in the figure 7.1 by 3, is updated at ever step of the protocol. The protocol develops interactively, as the user clicks buttons such as the ones denoted by 1 and 2. As each step of the protocol is completed, further explanations are made available to the user. This is more easily visible in Figure 7.2:

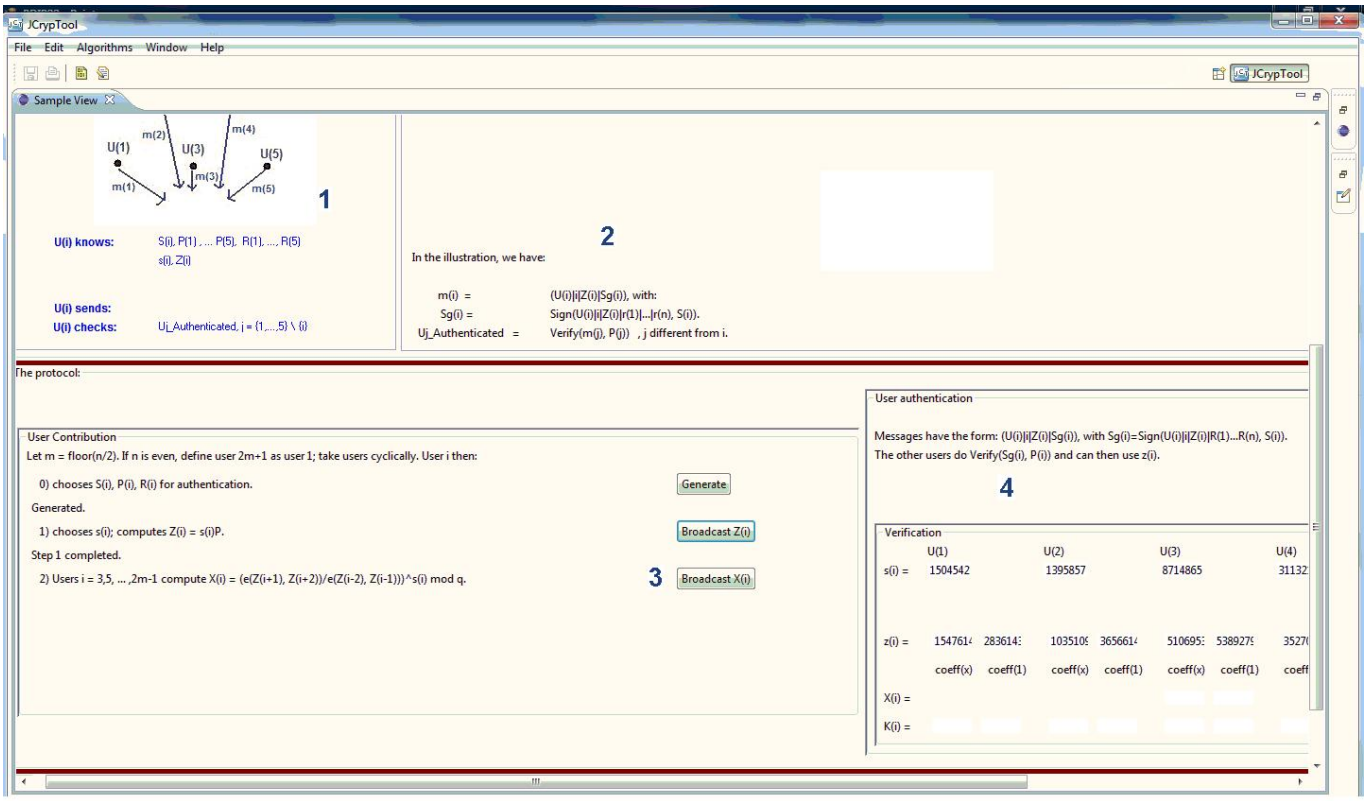

Figure 7.2: Protocol Information and User Interaction

The illustration panel 1 is updated with protocol information; explanations related to this information are written in the adjacent panel 2. The protocol can be continued by the user's interaction with the protocol panel 3. The authentication and verification panel 4 provides additional explanations regarding each of the protocol steps and also shows the user data. For $n = 4, 5$, or 6, the verification panel shows the values of s_i, z_i, X_i, and K_i, as defined by section 5.2.1. These values can be used for verification only; changing or deleting a value will not affect the protocol. For a number of users greater than 6, only the user information of the first six users is shown in the verification panel.

We show the authentication and verification panel in more detail in Figure 7.3:

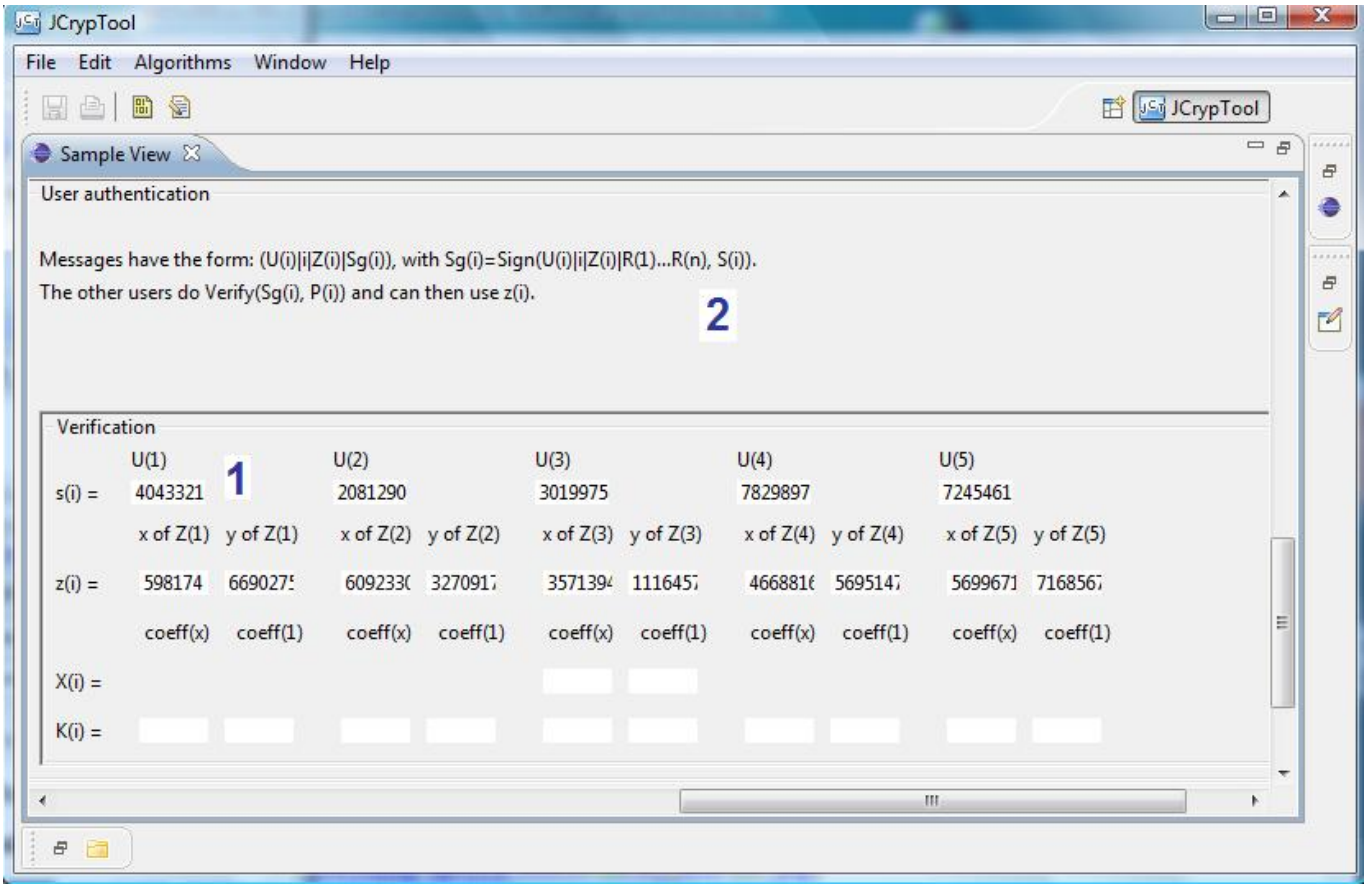

Figure 7.3: The User Authentication and Verification Panel

We point out in particular the verification panel 1, where the user information is displayed. The authentication panel 2 shows the authentication-related steps connected to each step of the protocol. For small numbers of users, it is visible from the verification panel that the keys computed by each user are identical. For larger numbers of users, however, only the user data of a few users is available. An automatic verification is provided by the interface as shown in figure 7.4 (the verification step is denoted 2). Once the keys are all verified to be identical, the user is given two rerun choices: with a new number of users, and with new setting parameters. The rerun panel is denoted below by 3.

At each step the user interface provides the user with only that information which is currently relevant. As an example, we show a fragment of the interface before the choice of setting parameters; any remaining panels – such as the protocol, illustration, verification, and rerun panels – are hidden at this point.

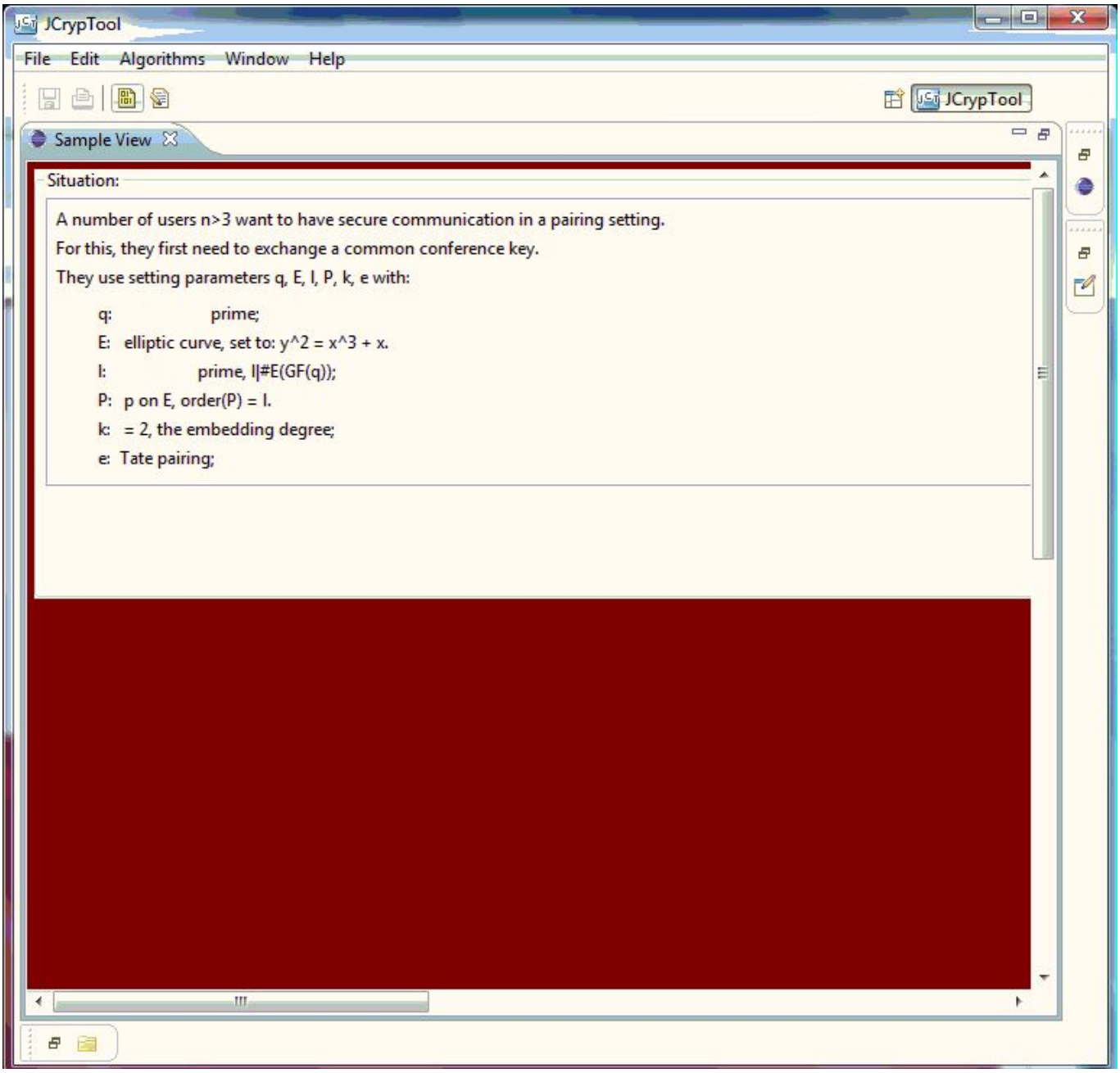

Figure 7.4: Initial Setting Description

7.3 Further Developments

As a short term goal, the key exchange component in JCrypTool is required to have the interfaces corresponding to the improvements mentioned in section 7.1. Apart from these modifications, the following extra items are listed for implementation:

- The implementation of the Weil pairing and adding it to the tripartite key exchange protocol (protocol-related).
- The implementation of the tripartite, BD I, and BD II protocols for pairings for embedding degree $k = 12$, with the aid of Paulo Barreto's library. (protocol-related, interface-related)
- Including a slow curve (a pairing friendly elliptic curve E for which denominator elimination is not applicable in Miller's algorithm – see chapter 2) as parameter of the $k = 2$ tripartite key exchange case. (protocol-related, interface-related)
- The implementation of turn-based BD I in finite fields and with pairings. (protocol-related, interface-related)
- Optimisation of the code: introduction of efficient polynomial reduction and arithmetic, optimisation of pairing computation. (protocol-related)
- Maintenance work for any of the included components.

Chapter 8

Conclusions

Several topics were presented throughout this thesis. Three main cryptographic settings are considered: finite fields, elliptic curves, and pairings. The parallel between finite fields and elliptic curves is easy to draw, by relating field elements to points and multiplication to addition. The translation of the Diffie-Hellman key exchange protocol and of the hard problems in finite fields and on elliptic curves is quite straight-forward, once exponentiation is replaced by scalar multiplication.

Pairings are a more complicated structure in general. They exist on any elliptic curve, but are only efficiently computable on the so-called pairing-friendly curves. They depend on the value of the embedding degree k. For the case $k = 2$, one usually must consider also a distortion map, while for the more efficient BN curves, one must alter the protocols so that the second argument of the pairing is chosen from a group G_2 that is linearly independent from the group G_1 where the first argument is taken from.

Several multi-partite key exchange protocols have been shown in this paper, in various settings. The Diffie-Hellman bipartite protocol is shown as the base key exchange arrangement for finite field and elliptic curve key exchange. The tripartite key exchange protocol due to Joux is the corresponding unit key exchange arrangement for pairing settings. These arrangements can be extended to the BD I and BD II multi-partite key exchange protocols.

Two versions of the BD I protocol were presented in this paper: a broadcast and a turn-based version. While the broadcast version contains less communication rounds, each user must perform more calculations. On the other hand, the turn-based protocol depends on the existence of a steady connection between the users, and on the fact that each user will react immediately after the previous user has acted. In both broadcast and turn-based BD I in finite fields, the arrangement is circular, while for pairings, it is triangle-based. This paper shows that the turn-based protocol in the pairing setting requires less rounds than the turn-based protocol in the finite field setting; however, each round is more complicated when pairings are considered. Similarly, the triangle-based, layered arrangement for BD II in the pairing setting takes less rounds than the tree-based arrangement in finite fields, but the pairing computation slows the protocol down.

All the above mentioned protocols rely on the difficulty of the discrete logarithm and computational and decisional Diffie-Hellman problems in their respective settings. With

parameters of proper size, they can be shown to be secure against passive attacks. In order to make these protocols secure against active attacks, one needs to provide authentication. One way of doing this is to use the Katz Yung compiler; the compiler, however, has running time linear in the number of users, so it will not be to the advantage of the BD II protocol, which is logarithmic in the number of users.

The visualisation of the protocols in CrypTool aims to present various facets of key exchange, demonstrating which parameters can influence the running time of the considered protocols. Several settings are provided for each protocol, and authentication is provided every time by using the Katz Yung compiler.

Appendix

In this appendix, we give a few interesting details regarding the implementation of key exchange protocols in various settings. Miller's algorithm has already been given in section 3.3; some further tricks may help in the writing of a fast implementation.

A.1 Scalar Multiplication and Exponentiation with Windowing Methods

In finite fields, repeated multiplication represents exponentiation. On elliptic curves, repeated addition represents scalar multiplication. We consider the following setting: p is a (large) prime, and we denote $\mathbb{F}_p^*$ the multiplicative subgroup of $\mathbb{F}_p$. Suppose we wish to calculate g^a for some elements $g, a \in \mathbb{F}_p^*$.

The input of the algorithm is: the prime p, $g \in \mathbb{F}_p$, and a non-negative integer a, whose binary decomposition is $(a_{l-1}, \ldots, a_0)$. The efficiency of the method depends on a window size that can depend on the application, denoted w; the higher the window size, the faster the algorithm. However, as the window size increases, the precomputation necessary for the scalar multiplication increases, and more storage capacity is needed. If the element g is known from before the protocol is started, better methods may be used for scalar multiplication; once g is given during the protocol, however, each user can compute in parallel $g, g^3, \ldots g^{2^w-1}$. The output of the algorithm is the point g^a.

Scalar Multiplication on Finite Fields

Input: $g \in \mathbb{F}_p^*, a = (a_{l-1}, \ldots, a_0)_2 \in \mathbb{Z}, w \geq 1, \{g, g^3, \ldots, g^{2^k-1}\}$.
Output: $g^a \in \mathbb{F}_p$.

1. $b \leftarrow 1; i = l - 1;$
2. WHILE $i \geq 0$ DO:
   (a) IF $a_i = 0$ THEN: $b \leftarrow b^2; i \leftarrow i - 1;$
   (b) ELSE:

i. $s \leftarrow \max(i - w + 1, 0)$;
ii. WHILE $a_s = 0$ DO: $s = s + 1$;
iii. FOR $h = 1$ TO $i - s + 1$ DO $b \leftarrow b^2$;
iv. $u \leftarrow (a_i \dots a_s)_2$;
v. $b \leftarrow b \cdot g^u$;
vi. $i \leftarrow s - 1$;

3. RETURN: b.

We point out that this problem is essentially equivalent to having an elliptic curve E over a field of the form $\mathbb{F}_q$ for q a prime or prime power, and wishing to calculate nP for some non-negative integer n and a point $P \in E$. An algorithm similar to the one above can therefore be used for scalar multiplication on elliptic curves; this algorithm is presented for example in [1]. The run time of scalar multiplication can be improved by using signed window methods.

A typical value for the window size for elliptic curves $w = 3$. A much bigger window size, such as $w = 7$ will dramatically increase both the speed of the algorithm and its storage demands; the precomputation time will also be much larger in this case. In the case of finite fields, a larger window size is admitted. If the precomputed list of points $\{P, 3P, \dots, (2^w - 1)P\}$ and of field elements $\{g, g^3, \dots, g^{2^w-1}\}$ are given as public parameters of a protocol and do not have to be computed by each of the participants involved, then it is certainly worth using a larger value of w. if the computation needs to be performed by each user independently, the value of w should be kept small.

A.2 A Horner-like Trick

The BD I key exchange protocol, both in finite fields and on pairings, contains expressions of the form $a_1^n \cdot a_2^{n-1} \cdot \dots \cdot a_n$. If each of these exponentiations is done separately and then the multiplication is performed, the user must do n exponentiations and n multiplications in the finite field. This calculation, however, can be done much faster if one considers using a Horner-like method.

Firstly, let us briefly mention what Horner's method is. Consider a polynomial $p \in \mathbb{K}[x]$ for some field $\mathbb{K}$. This polynomial is assumed to have degree n and can be written as:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 \,. \tag{8.1}$$

Then the evaluation of this polynomial at some value $x = z$ can be done as follows:

Horner's Method for Polynomial Evaluation

Input: $p(x) = a_n x^n + \cdots + a_1 x + a_0, \quad x = z.$
Output: $p(z)$.

1. $b_n = a_n$;
2. FOR $k = n-1$ TO $k = 0$ DO: $b_k = a_k + z b_{k+1}$;
3. RETURN: b_0.

This method can be more easily visualised as follows: instead of computing $(((a_n x^n + a_{n-1}x^{n-1}) + a_{n-2}x^{n-2}) + \cdots + a_1 x) + a_0$, one computes: $(((a_n x + a_{n-1})x + a_{n-2})x + a_{n-3}) + \cdots + a_0$.

A method similar to this can calculate expressions of the form $a_1^n a_2^{n-1} \ldots a_n$. We show this algorithm below.

A Horner-like Multiplication Trick

Input: $a_1, \ldots a_n$.
Output: $a_1^n \cdot a_2^{n-1} \cdot \cdots \cdot a_n$.

1. $s = t = a_1$;
2. FOR $k = 1$ TO $k = n-1$ DO:
   (a) $s = s \cdot a_{k+1}$;
   (b) $t = s \cdot t$;
3. RETURN: t.

With the aid of this method, each user is able to calculate the result $a_1^n \cdot \cdots \cdot a_n$ with only $2n-2$ field multiplications. As exponentiation is much more expensive than field multiplication, this Horner method is much cheaper than straight-forward successive exponentiation and multiplication.

A.3 Montgomery's Trick

Throughout the paper, we repeat that inversions can be turned into multiplications. This can be done in a variety of ways. For example, on elliptic curve, one can choose to use Jacobian instead of affine coordinates to obtain inversion-free addition and scalar multiplication

on the elliptic curve. In the case of pairings, one may use the bilinearity property to compute $\hat{e}(P, Q)^{-1}$ as $\hat{e}(-P, Q)$. This is generally a good strategy, as inverting a point on an elliptic curve is almost a free operation.

If field multiplications are much faster than inversions, one can also use the so-called Montgomery trick. Given a and b, this method will compute a^{-1} and b^{-1}. We add that this algorithm need not be used exclusively with integers or integers modulo p; any field will do. We show the trick below:

Montgomery's Trick

Input: $a, b \in \mathbb{K}$, for some field $\mathbb{K}$.
Output: a^{-1} and b^{-1}.

1. $\pi = ab$;
2. $\overline{\pi} = \frac{1}{\pi}$;
3. RETURN: $a^{-1} = \overline{\pi} \cdot b$ and $b^{-1} = \overline{\pi} \cdot a$.

Clearly, this method is not worth applying for a single inversion. However, if one must invert two elements, one may trade one of these inversions for three multiplications. On most platforms and in most fields, inversions are much slower than multiplications, so this trick can be very effective.

Bibliography

[1] R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, F. Vercauteren, "Handbook of Elliptic and Hyperelliptic Curve Cryptography", Chapman & Hall CRC, 2005.

[2] E. Bach, J. Shallit, "Algorithmic Number Theory", vol. 1, The MIT press, 1997.

[3] R. Balasubramian, N. Koblitz, "The improbability that an elliptic curve has subexponential discrete log problem under the Menezes-Okamoto-Vanstone algorithm", J. Cryptology, 11 (1998), pages 141 – 145.

[4] P. S. L. M. Barreto, M. Naehrig, "Pairing-friendly Elliptic Curves of Prime Order", Proceedings of SAC 2005, LNCS, vol. 3897, 2006, pages 319 – 331.

[5] G. Birkhoff and S. MacLane, "A Survey of Modern Algebra, 5^{th} edition", New York, Macmillan, 1996.

[6] D. Boneh, M. Franklin, "Identity-Based Encryption from the Weil Pairing", SIAM J. of Computing, vol. 32, no. 3, pages 586 – 615, 2003.

[7] M. Burmester, Y. Desmedt, "A Secure and Efficient Conference Key Distribution System" LNCS vol. 950, pages 427 – 437, 1994.

[8] CrypTool, `http://www.cryptool.org` , September 18, 2008, 14:32.

[9] Y. Desmedt, T. Lange, "Revisiting pairing based group key exchange", Financial Cryptography and Data Security, 2008.

[10] Y. Desmedt, T. Lange, M. Burmester, "Scalable Authenticated Tree Based Group Key Exchange for Ad-Hoc Groups", LNCS vol. 4886, 1997, pages 104 – 118.

[11] T. Dierks, C. Allen, "The TLS Protocol Version 1.0", RFC Editor, 1999.

[12] W. Diffie, M. E. Hellman, "New Directions in Cryptography", IEEE Trans. on Info. Th., vol. IT-22, Nov. 1976, pages 644 – 654.

[13] ECRYPT Yearly Report on Algorithms and Keysizes (2007−2008), IST 2002−507932, D.SPA.28, 2008.

[14] Explicit Formulas Database, http://www.hyperelliptic.org/EFD/, September 18, 2008, 14:34.

[15] The FlexiProvider toolkit for the Java Cryptography Architecture: http://www.flexiprovider.de/, September 18, 2008, 10:34.

[16] S. D. Galbraith, F. Hess, F. Vercauteren, "Hyperelliptic Pairings", LNCS, vol. 4575, pages 108 − 131, Springer Berlin, 2007.

[17] D. R. Hankerson, A. J. Menezes, S. A. Vanstone "Guide to Elliptic Curve Cryptography", Springer, 2004.

[18] JCrypTool product page on SourceForge: http://sourceforge.net/projects/jcryptool/, September 18, 2008, 10:08

[19] Antoine Joux, "A One Round Protocol for Tripartite Diffie-Hellman", J. Cryptology, vol. 17/2004, pages 263 − 276.

[20] A. Joux and K. Nguyen, "Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups", J. Cryptology 16(4), 2003, pages 239−247.

[21] J. Katz, M. Yung, "Scalable protocols for authenticated group key exchange", Advances in Cryptology – Crypto 2003, vol. 2729 of LNCS, pages 110 − 125, Springer, 2003.

[22] D. E. Knuth, "The Art of Computer Programming", vol. 2: "Seminumerical Algorithms", Third Eddition, Addison-Wesley, 1997, exercise 3.1.6 b.

[23] A. J. Menezes, "Elliptic Curve Public Key Cryptosystems", Kluwer Academic Publishers, 1993.

[24] A. J. Menezes, P. C. van oorschot, S. A. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1996.

[25] V. Miller, "Uses of elliptic curves in cryptography", Advances in Cryptology - Crypto 1985, LNCS vol. 218 (1986), Springer-Verlag, pages 417 − 426.

[26] A. J. Menezes, T. Okamoto, S. A. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field," IEEE Tran. on Info. Th., vol. 39, pages 1639 − 1646, 1993.

[27] C. Onete, "Elliptic Curves and Pairing Based Cryptosystems", Internship report, Eindhoven University of Technology, 2008 .

[28] H. C. A. van Tilborg, "Fundamentals of Cryptology", Kluwer Academic Publishers, 2000.

[29] C. Zhao, F. Zhang, and J. Huang, "Efficient Tate Pairing Computation Using Double-Base Chains", 2006, http://citeseer.ist.psu.edu/zhao06efficient.html, September 18, 15:02.