New Trends in Elliptic Curve Cryptography

Christian Hanser
chhanser@gmail.com

Institute for Applied Information
Processing and Communications (IAIK)
Graz University of Technology
Inffeldgasse 16a
8010 Graz, Austria

Master Thesis

Supervisor: Dipl.-Ing. Dr.techn. Mario Lamberger

April, 2010

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

(Unterschrift)

Acknowledgements

First of all, I would like to thank my supervisor Mario Lamberger, who encouraged and inspired me to grapple with elliptic curves. He did not only point out the right direction but also had at any time a sympathetic ear for my concerns. I really enjoyed writing this thesis and found pleasure in dealing with the theory of elliptic curves. I am very thankful for the time and for the effort Mario spent on corrections and discussions, not to forget the patience Mario showed for my questions and the problems that came up.

Secondly, I would like to express my sincere thanks to my parents, Franz and Maria Hanser, who supported me in every possible way throughout the whole years of studying. Special thanks go to my girlfriend, Marlies Bergmann, who has always been there for me and has encouraged me as often as possible. Finally, I would like to thank my sister, Eva Hanser, who gave me some important pieces of advice concerning English grammar, especially concerning the sometimes tricky usage of tenses.

Abstract

Elliptic curves are of great importance in present-day cryptography. In the past two decades, much progress has been made to make elliptic curves ready for prime-time use. Especially in the past few years a lot of research has been undertaken in order to find alternative applications of elliptic curves and to make existing applications more efficient by arithmetic speedups.

In this master thesis we are going to discuss the Elliptic Curve Only Hash (ECOH), a hash algorithm based on elliptic curves, and Edwards curves, a new curve shape having a fast and complete addition law. As for ECOH, we are going take a look at its security properties, present mostly new attacks, which exploit ECOH's algebraic structure, and reveal the results of our memoryless version of the Ferguson-Halcrow second preimage attack. Then, in the last part of this thesis we are going give an introduction to Edwards curves. We are also going to analyze to which extent standardized elliptic curve can be converted to Edwards shape.

Keywords: ECOH, elliptic curves, Edwards curves, hash functions

Kurzfassung

Elliptische Kurven sind heutzutage ein wichtiger Bestandteil der Kryptografie. So wurden in den letzten zwei Jahrzehnten große Fortschritte gemacht, um elliptische Kurven alltagstauglich zu machen. Speziell in den letzten Jahren konnten viele alternative Anwendungen und arithmetische Geschwindigkeitsverbesserungen für elliptische Kurven gefunden werden.

In dieser Masterarbeit betrachten wir den Hashalgorithmus Elliptic Curve Only Hash (ECOH), welcher auf elliptischen Kurven basiert, als auch sogenannte Edwardskurven, eine neue Kurvenart mit einem vollständigen und schnellen Additionsgesetz. Was ECOH betrifft, so analysieren wir seine Sicherheit, stellen vorwiegend neue Attacken vor, welche ECOH's algebraische Struktur angreifen, und zeigen die Ergebnisse unserer speicherlosen Version der Ferguson-Halcrow Attacke. Im letzten Teil der Arbeit geben wir eine Einführung in die Theorie der Edwardskurven. Außerdem analysieren wir inwieweit sich standardisierte Kurven in Edwardsform bringen lassen.

Stichwörter: ECOH, elliptische Kurven, Edwards Kurven, Hashfunktionen

Contents

Chapter 1

Introduction

In 1985, Neal Koblitz [Kob87] and Victor S. Miller [Mil85] both discovered independently that elliptic curves over finite fields can be used for public key cryptography. They published a Diffie-Hellman type protocol that relies on elliptic curve groups.

Both of these two groundbreaking works were inspired by a paper of Hendrik Lenstra, who was the first who had found an application of elliptic curves in cryptography. In 1984, he introduced a new integer factoring method based on elliptic curves [Len87].

The security of elliptic curve cryptography (ECC) is based on the associated discrete logarithm problem (DLP). Due to the absence of a subexponential time algorithm for calculating discrete logarithms on elliptic curves, it is assumed that this problem is harder than the integer factorization problem (IFP), which is the foundation for other public key methods like RSA.

The hardness of the elliptic curve discrete logarithm problem (ECDLP) allows us to use shorter key lenghts compared to RSA. So, for example encryption with a 160 bit elliptic curve key provides more or less the same security as an 1024 bit RSA encryption [LV99].

Due to the reduced key lengths elliptic curves are especially suitable for environments with limited processing power and memory resources. Therefore they find application in smart cards, cell phones, PDAs and the like.

However, in 1991 the Weil pairing attack [MOV91] was discovered, which allows an efficient computation of the discrete logarithm on a certain type of curves, the so-called supersingular curves. This was a major impact since these curves had been used very often as they are easy to compute with.

This discovery and the failure of the closely related hyperelliptic curve cryptography [Kob89, ADH94] shattered the trust in elliptic curve cryptography for almost a decade. Additionally, this distrust was stirred up by the RSA fraction, who claimed that elliptic curves were not ready for widespread use and did not have the same well studied mathematical background that RSA had at that point of time.

At the end of the 1990s the trust in the security of elliptic curve cryptosystems could slowly be re-established as no more security flaws had been discovered and industrial standards bodies supported ECC. A reason for this was also NSA's support for ECC [KKM08].

Over time more and more cryptographic algorithms, such as DSA and El-Gamal were adapted to work with elliptic curves and also new applications, such as pairing based cryptography, which is a form of identity based cryptography, were found for it [SOK00, Jou00].

In this master thesis we are going to deal with a new elliptic curve hashing algorithm, the so-called Elliptic Curve Only Hash (ECOH) [BACS08], and with arithmetic speedups

that can be achieved through a new, special type of curves, called Edwards curves [Edw07].

Edwards curves provide a complete addition law, which means that the sum of two arbitrary points always yields a point on the curve. This is an important advantage over ordinary elliptic curves in Weierstrass form, where we have separate addition laws for the sum of two distinct points and the double of a point. Furthermore, Edwards curves have good arithmetic characteristics and outperform most other types of curves.

ECOH resembles in many ways Micciancio's and Bellare's MuHASH [BM97]. We are going to analyze ECOH's algorithm and its properties and since ECOH was participating in NIST's currently running SHA-3 contest, we are going to examine its security and describe some new and some already known attacks against ECOH's prehash. Thus, we are going to investigate quantum attacks as well as ordinary attacks that amongst others arise from various algebraic group properties.

1.1 Outline of this Thesis

Part I of this thesis is comprised of three chapters and is dedicated to give a comprehensive introduction to both ECC and hash functions. Chapter 2 covers algebraic preliminaries, the basic theory of algebraic curves and then develops the fundamental concepts of the theory of elliptic curves on it. In Chapter 3 we are going to outline basic concepts of hashing, deal with hash collision search techniques and sketch MuHASH. Chapter 4 extends on Chapter 2 and introduces the reader to the usage of elliptic curves in cryptography. Amongst others, it deals with the historical background of ECC, compares ECC with its main competitor RSA, points out the security properties of different curve types and sketches several state-of-the-art attacks against ECC.

Part II, which consists of Chapter 5, is devoted to the elliptic curve hashs ECOH and ECOH^2. We present several new attacks against ECOH as well as one sophisticated second preimage attack for which ECOH is considered to be broken. Moreover, we are going to show a memoryless variant of this attack and present two colliding preimages of a weakened version of ECOH operating on a smaller curve.

Part III is comprised of Chapter 6 and deals with arithmetic speedups achieved through Edwards curves. We take a deeper look at Edwards curves and twisted Edwards curves over prime fields as well as at Edwards curves over binary fields. At last, we are going to discuss the possibilities of converting standardized curves to Edwards form.

Finally, in Chapter 7 we summarize and conclude this thesis.

Part I

Preliminaries

Chapter 2

Introduction to Elliptic Curves

This chapter treats the mathematical fundamentals that are needed for the remainder of this master thesis. It briefly discusses the basics of algebra, algebraic geometry and finally it gives a short and general introduction to elliptic curves. The general parts of this chapter are mainly based on lecture notes [TF03, Fri05, Let07]. The remainder relies on various books [Sil92, BSS99, Eng99, CF05] and on some other works [Kop09, OKS00, Duq04]. You can safely skip this chapter, when you are already familiar with algebra and elliptic curves.

2.1 Basics of Group and Field Theory

This section gives a short introduction into the basics of group theory and finite fields. Both topics are connected and are a crucial part of ECC as well as public key cryptography in general.

2.1.1 A Short Introduction to Group Theory

A group is a basic structure in both algebra and number theory. Groups are used to abstract calculations with specific numbers and form the basis for many other algebraic concepts.

Definition 2.1. A *group* $(G, \cdot)$ is a set of elements G plus an arithmetic operation $\cdot$ satisfying the subsequent properties:

- The product of two elements $x, y \in G$: $x \cdot y$ yields always another element of G. In other words, G is closed with respect to $\cdot$.
- For all elements $x, y, z \in G$ holds: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$. *(associativity)*
- There is a neutral element $e \in G$, which means that $x \cdot e = e \cdot x = x$ for every $x \in G$. *(neutral element)*
- For each $x \in G$ there is an inverse element x^{-1}, such that: $x \cdot x^{-1} = x^{-1} \cdot x = e$. *(inverse element)*

A group $(G, \cdot)$ is called *abelian*, if it is commutative:

$$\forall x, y \in G : x \cdot y = y \cdot x$$

Groups can be also written in an additive way. Then, the operation $\cdot$ is replaced by $+$. As we are going to see later, this is the case for elliptic curve groups.

Example 2.1. These two groups are often used in algebra:

- $(\mathbb{Z}_m = \{0, \ldots, m-1\}, +)$ forms an additive, abelian group. Addition is done modulo m, 0 is the neutral element and each element x has the inverse $-x$.
- Let p be prime. Then, $(\mathbb{Z}_p^* = \{1, \ldots, p-1\}, \cdot)$ is a multiplicative, abelian group modulo p with neutral element 1.

Groups $(G, \cdot)$ that allow every element $y \in G$ to be expressed as a power of another element $g \in G$ (i.e. $y = g^k$ with $k \in \mathbb{Z}$) are called *cyclic* and g is the so-called *generator* of the group. In such a case we write $G = \langle g \rangle$. Similarly, we can write every element y of a cyclic additive group $(G', +)$ with generator g as a multiple of g: $y = k \cdot g$.

Example 2.2. The generators of the group $(\mathbb{Z}_5^*, \cdot)$ are 2 and 3:

$g = 2$	$g = 3$
$2^1 = 2$	$3^1 = 3$
$2^2 = 4$	$3^2 = 9 \equiv 4 \bmod 5$
$2^3 = 8 \equiv 3 \bmod 5$	$3^3 = 27 \equiv 2 \bmod 5$
$2^4 = 16 \equiv 1 \bmod 5$	$3^4 = 81 \equiv 1 \bmod 5$

The *order* of an element x, denoted by $ord_G(x)$, is the smallest, positive integer k, so that $x^k = e$ (or additively: $k \cdot x = e$). If there is no such k, then x is said to have infinite order. The order of generators equals the group's cardinality, which is also called the group order $ord(G)$ or alternatively $|G|$.

Example 2.3. The order of 4 in $(\mathbb{Z}_5^*, \cdot)$ is 2:

$$4^1 = 4,\ 4^2 = 16 \equiv 1 \bmod 5.$$

A *subgroup* $(H, \cdot)$ of some group $(G, \cdot)$ (denoted by $(H, \cdot) \leq (G, \cdot)$) is itself a group and H is a subset of G. Furthermore, there are some special kinds of subgroups, such as *torsion subgroups*:

Definition 2.2. Provided that $m | ord(G)$, $(G[m], \cdot)$ is called *torsion subgroup of order m* of the cyclic group $(G, \cdot)$. It contains each element whose order divides m:

$$G[m] = \{x \in G \mid x^m = e\}$$

Example 2.4. The torsion subgroup of order 2 of $\mathbb{Z}_5^*$ is $\mathbb{Z}_5^*[2] = \{1, 4\}$.

Group Homomorphisms

Generally speaking, a homomorphism is a linear, structure-preserving map between two algebraic structures. In the context of groups, a *homomorphism* f between two groups $(G, \cdot)$ and $(G', *)$ is:

$$f : G \to G'$$

such that for all $x, y \in G$:

$$f(x \cdot y) = f(x) * f(y)$$

holds. f also preserves the identity element, i.e. $f(e_G) = e_{G'}$.

A (group) homomorphism $f : G \to G'$ is called

- *monomorphism* if it is injective,
- *endomorphism* if it is surjective,
- *isomorphism* if it is bijective, and
- *automorphism* if it is bijective and $G = G'$.

Two groups G and G' are said to be *isomorphic*, if an isomorphism between them exists. This is denoted as $G \simeq G'$.

The Discrete Logarithm Problem

We already saw that in cyclic groups every element x can be expressed as a power of the generator g: $x = g^k$. When k is chosen to be the smallest, positive integer possible, it is called the *discrete logarithm (DL)* or sometimes also *index* of x with respect to g, written as $log_g(x)$.

Example 2.5. In $(\mathbb{Z}_5^*, \cdot)$ we have $log_3(4) = 2$.

Just like the ordinary logarithm, the discrete logarithm has some important features:

- $\log_h(x) = \log_g(x) \cdot \log_g(h)$, and
- $\log_g(x \cdot y) = \log_g(x) + \log_g(y)$,

where g and h are both generators of the same cyclic group G.

Definition 2.3 (DLP)**.** Given a generator g of the cyclic group $(G, \cdot)$ and an element x in G, the difficulty of finding an integer k such that:

$$x = g^k$$

is called the *discrete logarithm problem (DLP)*.

Note that the hardness of the DLP depends on the group structure. So, for example it is hard in groups of the type $(\mathbb{Z}_p^*, \cdot)$ but not in groups of the shape $(\mathbb{Z}_p, +)$.

2.1.2 A Brief Glimpse into Field Theory

The notion of fields can be built on top of the group concept. The big difference to a group is that a field has got two distinct operations:

Definition 2.4 (Field)**.** A *field* $(K, +, \cdot)$ consists of a set of elements K and two operations $+$ and $\cdot$, where

- $(K, +)$ is an additive abelian group with neutral element 0,
- $(K^*, \cdot)$ is a multiplicative abelian group with neutral element 1, and
- both groups are adjoined by distributivity:
  - $\forall x, y, z \in K : x \cdot (y + z) = x \cdot y + x \cdot z$ *(left-distributivity)*
  - $\forall x, y, z \in K : (x + y) \cdot z = x \cdot z + y \cdot z$ *(right-distributivity)*

A field K is said to be *finite*, if it has only finitely many elements.

The *characteristic* of a field K, $char(K)$, is defined as the smallest integer $n \geq 1$ such that $n \cdot 1 = 0$. If there is no such n, the characteristic is defined to be 0. In general, the characteristic is either 0 or a prime p. Furthermore, finite fields always have prime characteristic.

Example 2.6. Commonly used fields are for example:

1. $(\mathbb{R}, +, \cdot)$ is the infinite field of real numbers.
2. For some prime p, $(\mathbb{Z}_p, +, \cdot)$ is a finite field often used for cryptographic purposes. In future we will write $\mathbb{F}_p$ for it.

Field Homomorphisms

In the context of fields, a *homomorphism* f between two fields K and K' is:

$$f : K \to K'$$

such that for all $x, y \in K$:

$$f(x + y) = f(x) + f(y)$$

and

$$f(xy) = f(x) \cdot f(y)$$

holds. Additionally, we have $f(0_K) = 0_{K'}$, $f(1_K) = 1_{K'}$ and for all $x \neq 0$ we have $f(x^{-1}) = f(x)^{-1}$.

Two fields K and K' are said to be *isomorphic*, when an isomorphism between them exists. This is denoted as $K \simeq K'$.

Polynomials over Fields

Let K be any field. Then, the following set

$$K[X_1, \ldots, X_n] \coloneqq \left\{ f(X_1, \ldots, X_n) = \sum_{i_1, \ldots, i_n \geq 0} c_{i_1, \ldots, i_n} X_1^{i_1} \ldots X_n^{i_n} : c_{i_1, \ldots, i_n} \in K \right\}$$

with only finitely many non-vanishing coefficients $c_{i_1, \ldots, i_n}$ is called the *ring of polynomials over K in n variables*.

The *total degree* of each non-vanishing monomial $X_1^{i_1} \ldots X_n^{i_n}$ is $i_1 + \ldots + i_n$. The maximum total degree is called the polynomial's degree $\deg(f)$, if $f \neq 0$. Otherwise, $\deg(f)$ is defined to be $-\infty$. A polynomial $f(X) = c_n X^n + \ldots + c_1 X + c_0 \in K[X]$ is said to be *monic*, if its *leading coefficient* $c_n = 1$, *reducible*, if $f(X) = g(X) \cdot h(X)$ with $0 < \max\{\deg(g), \deg(h)\} < \deg(f)$ holds and *irreducible*, otherwise.

The Field of Rational Functions

The *field of rational functions* is a special case of a more general construction called the *field of fractions* and is defined to be:

$$K(X_1, \ldots, X_n) \coloneqq \left\{ f \cdot g^{-1} : f, g \in K[X_1, \ldots, X_n] \text{ with } g \neq 0 \right\},$$

where K is an arbitrary field.

Field Extensions and Finite Fields

An extension field L can be constructed from a base field K through the adjunction of new elements. In order to acquire new elements the ring of polynomials over K is factored by a monic, irreducible polynomial $f(X)$:

$$L = K[X]/(f(X)) := \left\{ \sum_{i=0}^{n} c_i X^i \bmod f(X) : c_i \in K, n \geq 0 \right\}$$

The result is a new field $L \supset K$ that is a vector space over K. This is denoted by L/K. The degree of $f(X)$ equals the *degree of the field extension* $[L : K]$ and is the dimension of the vector space.

A field L/K is said to be *algebraically closed over* K, if every non-constant polynomial in one variable $g \in K[X] \setminus K$ has a root in L. L is then also called the *algebraic closure* of K. Let α be a root of $f(X)$, then $L = K[X]/(f(X))$ is isomorphic to

$$K(\alpha) = K[\alpha] = \left\{ c_{n-1}\alpha^{n-1} + \ldots + c_1\alpha + c_0 : c_i \in K \right\}.$$

In further consequence this means that every element in L has a representation of the form $c_{n-1}\alpha^{n-1} + \ldots + c_1\alpha + c_0$, called *polynomial representation*. For elements of finite fields this implies a second representation, aside from the representation as a power of some generator. In order to speed up arithmetic operations the ordinary representation can be used for multiplication and the polynomial representation for the addition of two elements.

So far, the only finite fields we have considered were of the shape $\mathbb{F}_p = \mathbb{Z}_p$. But actually there is a finite field for every prime power:

Theorem 2.1. *For every prime power $q = p^n$ with $n \geq 1$, there exists exactly one finite field $\mathbb{F}_q$ with q elements (up to isomorphism).*

Every such finite field $\mathbb{F}_{p^n}$ comprises the subfield $\mathbb{Z}_p$. Note also that the multiplicative group $(\mathbb{F}_q^*, \cdot)$ of any finite field $\mathbb{F}_q$ is always cyclic, and so every element in $(\mathbb{F}_q^*, \cdot)$ is a power of some generator g.

Definition 2.5. An element $\xi \in \mathbb{F}_q$ is called *n^{th}-root of unity*, if $\xi^n = 1$ or in other words: if its order divides n. It is a primitive root of unity, if $\xi^k \neq 1 \ \forall k = 1, \ldots, n-1$. The set of all *$n^{th}$-roots of unity* is a subgroup of $(\mathbb{F}_q^*, \cdot)$ and is denoted by $\mu_n(\mathbb{F}_q)$.

So, for instance every generator of $(\mathbb{F}_q^*, \cdot)$ is a (primitive) root of unity.

Example 2.7. We are going to create the finite field $\mathbb{F}_4$. To do so we need a monic, irreducible polynomial: $f(X) = X^2 + X + 1 \in \mathbb{F}_2[X]$. Let α be a root of $f(X)$, which implies that

$$\alpha^2 = -(\alpha + 1) = \alpha + 1, \ \alpha^3 = \alpha^2 + \alpha = 2\alpha + 1 = 1.$$

Hence, $\mathbb{F}_4 = \mathbb{F}_2[\alpha] = \{0, \alpha, \alpha + 1, 1\} = \{0, \alpha, \alpha^2, \alpha^3\}$.

2.1.3 Finite Field Arithmetic

This subsection shows some identities and defines some crucial notions concerning finite field arithmetics.

Fermat's Little Theorem

Fermat's little theorem states that for a prime p we have $a^p \equiv a \bmod p$ for all $a \in \mathbb{Z}_p$. Likewise, its generalization to finite fields expresses that $a^q = a$ for all $a \in \mathbb{F}_q$.

The Frobenius Homomorphism

Unlike in fields with characteristic zero, in fields with prime characteristic p the term $(x+y)^p$ evaluates to $x^p + y^p$:

Lemma 2.1 (Freshman's dream)**.** *Fix a field K of prime characteristic p. Then, we have:*

$$(x+y)^p = x^p + y^p$$

Proof. This holds as:

$$(x+y)^p = \sum_{k=0}^{p} \binom{p}{k} x^k y^{p-k} = \sum_{k=0}^{p} \underbrace{\frac{p!}{k!(p-k)!}}_{(*)} x^k y^{p-k} =$$

p divides the term $(*)$, except for the cases $k = 0$ and $k = p$, in which the numerator and the denominator cancel each other out. This means that the following two terms remain:

$$= x^0 y^p + x^p y^0 = x^p + y^p.$$

□

That is why the map $\phi : K \to K$ satisfying $\phi(x) = x^p$ is a field homomorphism, that is

- $\phi(x+y) = (x+y)^p = x^p + y^p = \phi(x) + \phi(y)$
- $\phi(xy) = (xy)^p = x^p y^p = \phi(x) \cdot \phi(y)$

ϕ is called *Frobenius homomorphism*. It is an automorphism if K is finite.

Legendre Symbol

The Legendre symbol signifies, whether some element $a \in \mathbb{F}_p$ is a square modulo p.

Definition 2.6 (Legendre symbol)**.** Fix some prime $p \geq 3$. Then, the *Legendre symbol* is defined as follows:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a \equiv 0 \bmod p, \\ 1 & \text{if } a \not\equiv 0 \bmod p \text{ and } \exists x : x^2 \equiv a \bmod p, \\ -1 & \text{if there is no such } x. \end{cases}$$

If

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & a \text{ is called } \textit{quadratic residue} \pmod p, \\ -1 & a \text{ is called } \textit{quadratic non-residue} \pmod p. \end{cases}$$

The congruence below shows how to evaluate its value:

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \bmod p$$

There exist also several generalizations of the Legendre symbol. The one below works for fields $\mathbb{F}_q$ with $q = p^n$ and $n \in \mathbb{N}$.

$$\chi(x) = \begin{cases} 0 & \text{if } x = 0, \\ 1 & \text{if } \exists y \in \mathbb{F}_q : x = y^2, \\ -1 & \text{otherwise.} \end{cases}$$

The Trace Function

The trace function plays an important role when it comes to solving equations in binary finite fields.

Definition 2.7 (Trace function)**.** The *trace function* $Tr : \mathbb{F}_{2^n} \to \mathbb{F}_2$ is defined by

$$Tr(x) = \sum_{i=0}^{n-1} x^{2^i}.$$

Proposition 2.1 (Properties of the trace function)**.** *Let* $x, y \in \mathbb{F}_{2^n}$.

1. $Tr(x + y) = Tr(x) + Tr(y)$,
2. $Tr(x) = Tr(x^2) = Tr(x)^2$,
3. $Tr(x) \in \{0, 1\}$.

Proof. In $\mathbb{F}_{2^n}$, the Frobenius homomorphism is $\phi : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ with $\phi(x) = x^2$.

1. This is obviously true as the trace function can be rewritten as follows:

   $$Tr(x) = x + \phi^1(x) + \phi^2(x) + \ldots + \phi^{n-1}(x)$$

   and $\phi^k(x) = x^{2^k}$ in binary fields.
2. Clearly, $Tr(x)^2 = Tr(x^2) = x^2 + x^4 + x^8 + \ldots + x^{2^{n-1}} + \underbrace{x^{2^n}}_{x} = Tr(x)$.
3. This follows from $Tr(x)^2 = Tr(x)$.

□

Solving Quadratic Equations in $\mathbb{F}_{2^n}$

In $\mathbb{F}_{2^n}$, a quadratic equation has the shape

$$X^2 + aX + b = 0, \tag{2.1}$$

where we assume $a \neq 0$, because otherwise we would compute $X^2 = b$ whose only solution is $x_0 = \sqrt{b} = b^{2^{n-1}}$. Now, we apply the change of variables $X \leftarrow X \cdot a^{-1}$ and get the equation

$$X^2 + X = c \text{ with } c = b \cdot a^{-2}.$$

This equation has solutions x_1 and $x_1 + 1$ if and only if $Tr(c) = 0$, where

$$x_1 = \begin{cases} \sum_{i=0}^{\frac{n-3}{2}} c^{2^{2i+1}} & \text{if } n \text{ is odd,} \\ \sum_{i=0}^{n-1} \left(\sum_{j=0}^{i} c^{2^j} \right) y^{2^i} & \text{otherwise} \end{cases}$$

with $y \in \mathbb{F}_{2^n}$ having trace 1. Apparently, $a \cdot x_1$ and $a \cdot (x_1 + 1)$ are then solutions of Equation (2.1).

2.2 The Affine and the Projective Space

When we are talking about elliptic curves and algebraic geometry in general it is essential to know what is meant by the notions of the affine and the projective space.

Definition 2.8. Let K be an arbitrary field. Then, the *affine space* K^n over K is the set of all n-tuples $(x_1, x_2, \ldots, x_n)$ with $x_i \in K \ \forall i = 1, \ldots, n$.

Definition 2.9. Let K be an arbitrary field. The *projective space* $\mathbb{P}^n(K)$ over K is the set of all $(n+1)$-tuples $(x_1, x_2, \ldots, x_{n+1}) \neq 0$ with $x_i \in K \ \forall i = 1, \ldots, n+1$. On these $(n+1)$-tuples we have the following equivalence relation:

$$(x_1, x_2, \ldots, x_{n+1}) \equiv (x'_1, x'_2, \ldots, x'_{n+1})$$

if there is a $0 \neq \lambda \in K$ such that $x_i = \lambda x'_i \ \forall i = 1, \ldots, n+1$. We write $(x_1 : x_2 : \ldots : x_{n+1})$ for *projective points*.

This means nothing more than identifying every line that goes through the origin with a single projective point. In this way, an $(n+1)$-dimensional affine space is scaled down to n dimensions. The projective space has some major advantages over the affine space. For our purposes it is important that we are able to define points at infinity and that inversions of field elements can be avoided, which helps saving precious computation time.

2.2.1 Relation between Affine and Projective Coordinates

We can switch between affine and projective coordinates in the subsequent ways:

- In order to map an affine point $(x_1, \ldots, x_n)$ to its projective counterpart we take an arbitrary point $0 \neq z \in K$ and compute $(x_1 z : x_2 z : \ldots : x_n z : z)$.
- Conversely, we map the projective point $(x_1 : x_2 : \ldots : x_n : z)$ to the affine point $(x_1/z, \ldots, x_n/z)$ provided that $0 \neq z \in K$.

In our future considerations we are mostly going to deal with the projective plane $\mathbb{P}^2(\mathbb{F}_q)$.

2.3 Algebraic Curves

When we are talking about curves in analysis we usually define them through continuous parametrizations. In algebraic geometry, however, neither continuous functions nor parameterizations do exist. Therefore, we must choose another approach:

Definition 2.10. Let K be any field, $\overline{K}$ be an algebraic closure of K and let f be an (irreducible) polynomial in $K[X_1, X_2, \ldots, X_n]$. Then, an *affine algebraic curve* $C_f(K)$ is the set

$$\{(x_1, x_2, \ldots, x_n) \in K^n : f(x_1, x_2, \ldots, x_n) = 0\}.$$

We usually denote the curve $C_f(\overline{K})$, which is a superset of $C_f(K)$, by C_f.

This means that curves are defined by polynomials and can be written as equations in n variables. The subsequent definitions are important with respect to algebraic curves:

Definition 2.11. Let K be any field and $\overline{K}$ be an algebraic closure of K. Furthermore, let $f = \sum_{i_1,\ldots,i_n \geq 0} c_{i_1,\ldots,i_n} X_1^{i_1} \ldots X_n^{i_n} \in K[X_1, \ldots, K_n] \setminus K$ be a polynomial and $P = (x_1, \ldots, x_n)$ be an arbitrary point in $C_f(\overline{K})$.

- The *order* of a curve C_f at point P is defined by:

  $$ord_P(C_f) = \min\{i_1 + \ldots + i_n : \tilde{c}_{i_1,\ldots,i_n} \neq 0\}.$$

  where the coefficients $\tilde{c}_{i_1,\ldots,i_n} \in \overline{K}$ originate from the *Taylor series of* f evaluated at point P:

  $$\sum_{i_1,\ldots,i_n \geq 0} \tilde{c}_{i_1,\ldots,i_n} (X_1 - x_1)^{i_1} \ldots (X_n - x_n)^{i_n}$$

- P is said to be *regular* if and only if $ord_P(C_f) = 1$ and *singular* otherwise. Moreover, a curve C_f containing only regular points is called *smooth* or *regular*.

- Let $K \subseteq \tilde{K} \subseteq \overline{K}$ be a field. P is called $\tilde{K}$*-rational*, if $x_1, \ldots, x_n \in \tilde{K}$.

- $K[C_f] = K[X_1, \ldots, X_n]/(f)$ is called the *coordinate ring of* C_f and is defined as

  $$\{g \bmod f : g \in K[X_1, \ldots, X_n]\}.$$

- If f is irreducible in $K[X_1, \ldots, X_n]$, then $K(C_f) = K(X_1, \ldots, X_n)/(f)$, which is defined as

  $$\left\{(g \bmod f) \cdot (h \bmod f)^{-1} : g, h \in K[X_1, \ldots, X_n] \text{ with } h \not\equiv 0 \bmod f\right\}$$

  exists and is said to be the *function field of* C_f.

2.3.1 Projectively Closed Curves

Polynomials can be represented either affinely or projectively. Homogenization allows us to switch a polynomial's representation from affine to projective:

Definition 2.12. Let f be a polynomial of degree d. We split f into homogeneous polynomials f_i, such that every f_i has total degree i: $f(X_1, \ldots, X_n) = \sum_{i=1}^{d} f_i(X_1, \ldots, X_n)$. Now, we define the *projective closure* $\bar{f}$ of f through homogenization:

$$\bar{f}(X_1, \ldots, X_n, Z) = \sum_{i=1}^{d} f_i(X_1, \ldots, X_n) Z^{d-i}$$

The result is a polynomial comprising only monomials of total degree d. Moreover, it turns out that each affine point $(x, y) \in C_f$ is embedded into the projective curve $C_{\overline{f}}$ by the point $(x : y : 1)$. Since we have got an extra variable Z, this curve may contain additional points.

This brings us to the following definitions:

Definition 2.13. We are given a polynomial f and its projective closure $\overline{f}$.

1. The curve $C_{\overline{f}}$ is called the *projective closure* of C_f.
2. A point of the shape $(x : y : 0) \in C_{\overline{f}}$ is called *point at infinity*. The set of all those points is named *line at infinity*.

Example 2.8. Let $f(X, Y) = X^3 + XY + Y \in \mathbb{Z}_3[X, Y]$. After homogenization we obtain a new polynomial $\bar{f}(X, Y, Z) = X^3 + XYZ + YZ^2 \in \mathbb{Z}_3[X, Y, Z]$. The curve $C_f(\mathbb{Z}_3)$ comprises the points $(0, 0)$ and $(1, 1)$, whereas $C_{\overline{f}}(\mathbb{Z}_3)$ consists of their projective counterparts $(0 : 0 : 1), (1 : 1 : 1)$ as well as several points at infinity, namely $(0 : 0 : 0), (0 : 1 : 0)$ and $(0 : 2 : 0)$.

2.4 Elliptic Curves

Now we are ready to introduce the notion of an elliptic curve:

Definition 2.14 (Elliptic curve)**.** Let K be any field. An *elliptic curve* E over the field K, denoted by E/K, is a plane, smooth curve described by the subsequent affine equation:

$$\boxed{Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6} \tag{2.2}$$

or alternatively by the equivalent projective equation:

$$\boxed{Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3} \tag{2.3}$$

where $a_1, \ldots, a_6 \in K$. Equation (2.2) is called long Weierstrass equation. The set of all points $(x, y) \in K^2$ satisfying Equation (2.2) plus $\mathcal{O} = (0 : 1 : 0)$, the point at infinity, is denoted by $E(K)$.

Note that K is an arbitrary field, it can be either finite or infinite.

2.4.1 Summation of Points and the Group Laws

By now, it has not been quite obvious how elliptic curves can be utilized for data encryption. In order to perform computations on the set $E(K)$ we have to define an arithmetic operation $+$. Before we do so, we need to know the following facts:

Proposition 2.2. *When we draw a line L through two distinct points $P, Q \in E(K)$ this line always intersects at a third point $R \in E(K)$. The same holds, when we lay a tangent L through a point $P \in E(K)$. Then, this tangent also intersects at a third point $R \in E(K)$ (if we count P twice).*

Proposition 2.3. *The reflection $\overline{P}$ of a point $P = (x, y) \in E(K) \setminus \{\mathcal{O}\}$ on the curve is given by $\overline{P} = (x, -y - a_1x - a_3)$.*

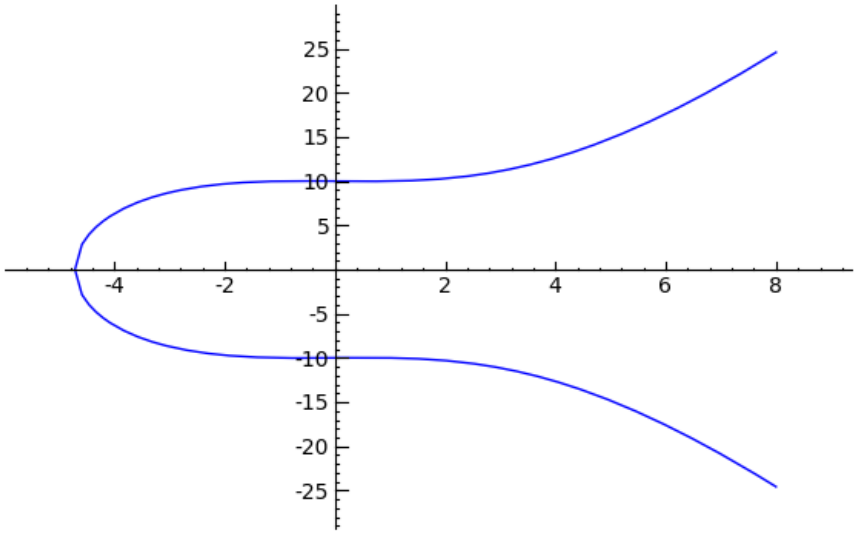

Figure 2.1: Plot of the curve $Y^2 = X^3 - X + 100$ over $\mathbb{R}$

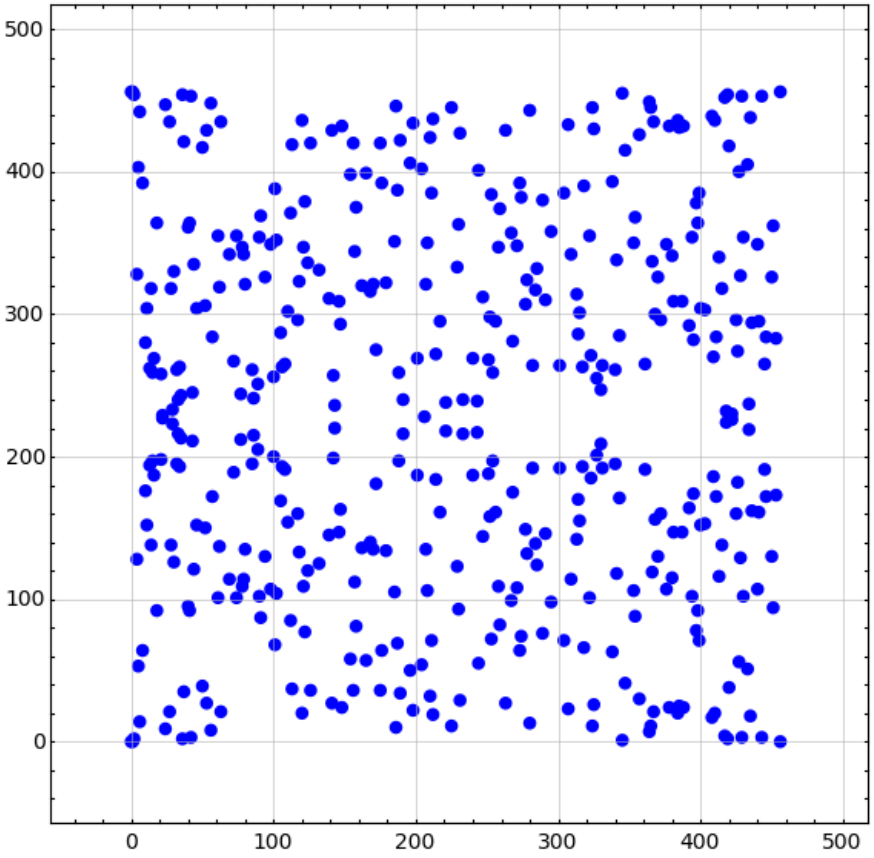

Figure 2.2: Plot of the curve $Y^2 + Y = X^3 - X$ over $\mathbb{F}_{457}$

For the addition law, we distinguish two different cases:

1. Let $P, Q \in E(K)$ be two distinct points. In order to sum P and Q we lay a line through both points. This line intersects at a third point $R = (x_R, y_R) \in E(K)$. Then, the sum $P + Q$ is defined by $\overline{R}$.

2. Let $P \in E(K)$. For doubling P we draw a tangent through P, which usually intersects at a third point $R \in E(K)$. If this is not the case, we set $R = \mathcal{O}$. Again, we obtain $2P = P + P$ by reflecting R, i.e. $2P = \overline{R}$.

These two summation instructions are known as *chord-method* and *tangent-method* and are valid for both finite and infinite fields K. The exact formulae are being detailed by Algorithm 2.1. Note that these formulae simplify according to the type of curve and the characteristics of the field K.

Together with the set $E(K)$ the addition laws form an abelian group $(E(K), +)$, which means that the following group laws hold:

1. $E(K)$ is closed under the operation $+$, which means that the sum of two arbitrary points always yields a third point on the curve.

2. It is associative:

$$\forall P, Q, R \in E(K) \text{ we have: } (P + Q) + R = P + (Q + R)$$

3. It is commutative:

$$\forall P, Q \in E(K) \text{ holds: } P + Q = Q + P$$

4. There exists a neutral element $\mathcal{O}$, so that:

$$\forall P \in E(K) : P + \mathcal{O} = \mathcal{O} + P = P$$

5. Every point P in $E(K)$ has an inverse $-P \in E(K)$, namely its reflection $\overline{P} = -P$, such that

$$P + (-P) = (-P) + P = \mathcal{O}$$

2.4.2 The Discriminant and the j-Invariant

The discriminant and the j-invariant are essential properties of an elliptic curve. They are made up as follows:

Definition 2.15 (Discriminant and j-invariant)**.** Let K be any field and E/K be an elliptic curve and

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

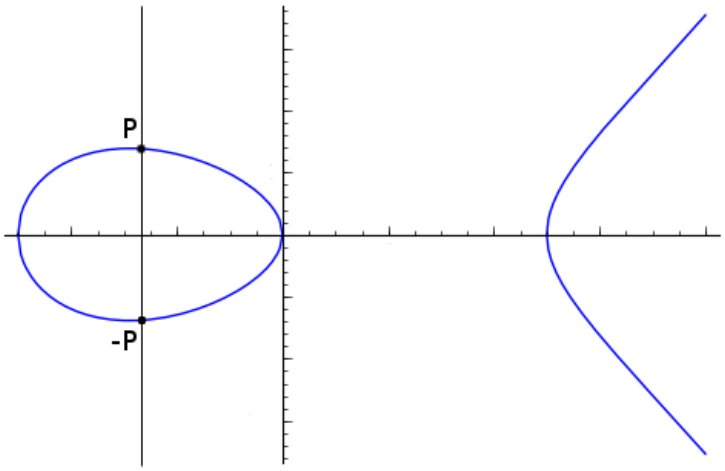

Figure 2.3: The inversion of point P on the curve $Y^2 = X^3 - 25X$ over $\mathbb{R}$

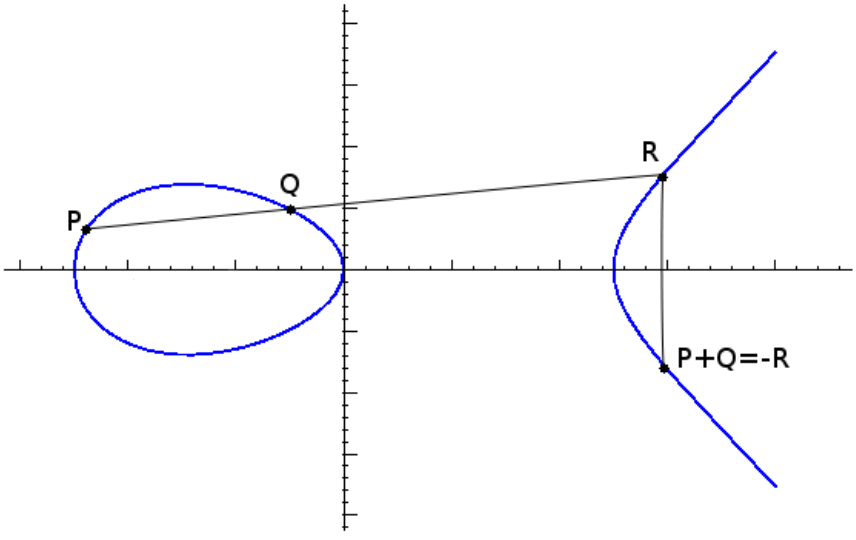

Figure 2.4: Addition of points P and Q on the curve $Y^2 = X^3 - 25X$ over $\mathbb{R}$

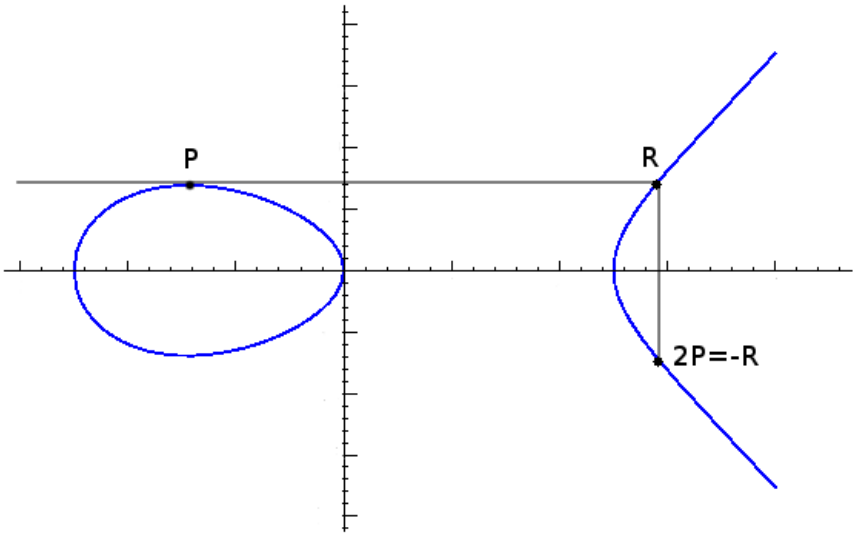

Figure 2.5: Doubling of point P on the curve $Y^2 = X^3 - 25X$ over $\mathbb{R}$

Algorithm 2.1 Addition of two points on an elliptic curve in Weierstrass form

Require: $P, Q \in \mathbb{E}(K)$

Ensure: $R \in E(K)$

1: Check whether $Q = -P$. If this is the case $P + Q$ equals $\mathcal{O}$. Set $R = \mathcal{O}$ and go to 3.

2: Otherwise, compute $P + Q = R = (x_3, y_3) \in E(K) \setminus \{\mathcal{O}\}$, where

- $P = (x_1, y_1)$,
- $Q = (x_2, y_2)$,
- $x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2$, and
- $y_3 = -(\lambda + a_1)x_3 - \mu - a_3$.

So far, everything applies for both $Q = P$ and $Q \neq P$. This distinction is contained in the values λ and μ:

$$\lambda = \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1} & \text{if } x_1 \neq x_2, \\ (3x_1^2 + 2a_2x_1 + a_4 - a_1y_1)(2y_1 + a_1x_1 + a_3)^{-1} & \text{otherwise.} \end{cases}$$

and

$$\mu = \begin{cases} (y_1x_2 - y_2x_1)(x_2 - x_1)^{-1} & \text{if } x_1 \neq x_2, \\ (-x_1^3 + a_4x_1 + 2a_6 - a_3y_1)(2y_1 + a_1x_1 + a_3)^{-1} & \text{otherwise.} \end{cases}$$

3: **return** R.

Then

- $\boxed{\Delta(E) = -b_2^2 b_8 - 8b_4^3 - 27b_6^2 + 9b_2 b_4 b_6}$ is called *discriminant*, and
- $\boxed{j(E) = (b_2^2 - 24b_4) \cdot \Delta(E)^{-1}}$, for $\Delta(E) \neq 0$ is called *j-invariant*.

The importance of the discriminant is that $\Delta(E)$ is non-zero if and only if the curve E is non-singular. A non-zero discriminant ensures that the polynomial $X^3+a_2X^2+a_4X+a_6$ has no multiple roots in K and hence that any point on the curve has a uniquely determined tangent line.

The definition of the j-invariant brings us to the subsection below.

2.4.3 Simplified Curve Equations

The curve equations simplify depending on the j-invariant and the characteristic of the underlying field K:

Proposition 2.4. *Let E/K be an elliptic curve in long Weierstrass form with coefficients $a_1, \ldots, a_6$. Under the assumptions below, there is a change of variables taking E into form E':*

- *If $char(K) = 2$:*
  - *If $j(E) = 0$:*

$$\boxed{E' : Y^2 + a_3 Y = X^3 + a_4 X + a_6} \tag{2.4}$$
$$j(E') = 0, \Delta(E') = a_4^3$$

  - *If $j(E) \neq 0$:*

$$\boxed{E' : Y^2 + XY = X^3 + a_2 X^2 + a_6} \tag{2.5}$$
$$j(E') = a_6^{-1}, \Delta(E') = a_6$$

- *If $char(K) \notin \{2, 3\}$:*

$$\boxed{E' : Y^2 = X^3 + a_4 X + a_6} \tag{2.6}$$
$$j(E') = 1728 \cdot 4a_4^3 \cdot (4a_4^3 + 27a_6^2)^{-1}, \Delta(E') = -16(4a_4^3 + 27a_6^2)$$

Proof. We refer to [Sil92, Proposition A.1.1]. □

E' is called *short Weierstrass form*. For fields K with $char(K) \notin \{2, 3\}$ also the addition, doubling and point negation formulae simplify. They are summarized in Table 2.1.

2.4.4 Mappings between Elliptic Curves

In the subsequent chapters, we must be able to switch between different curve shapes in order to speed up calculations. In the following we discuss several non-equivalent approaches.

$(x_1, y_1) + (x_2, y_2)$	$x_3 = \lambda^2 - x_1 - x_2$ $y_3 = \lambda(x_1 - x_3) - y_1$ $\lambda = \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1} & P_1 \neq P_2, \\ (3x_1^2 + a_4)(2y_1)^{-1} & \text{otherwise.} \end{cases}$
$-(x, y)$	$(x, -y)$

Table 2.1: Simplified addition formulae for $char(K) \notin \{2, 3\}$ and $(x_1, y_1) \neq -(x_2, y_2)$

Curve Isomorphisms and Twists

A curve isomorphism is a bijection between the sets of K-rational points $E(K)$ and $E'(K)$ of the two different curves E and E'.

Definition 2.16 (Isomorphic curves)**.** Two elliptic curves $E : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$ and $E' : Y'^2 + a'_1X'Y' + a'_3Y' = X'^3 + a'_2X'^2 + a'_4X' + a'_6$ defined over the same field K are *isomorphic* if and only if there are constants $u \in K^*$ and $r, s, t \in K$, such that the subsequent change of coordinates

$$X = u^2X' + r \text{ and } Y = u^3Y' + su^2X' + t$$

converts curve E into E'. This is denoted by $E(K) \simeq E'(K)$.

Elliptic curve isomorphisms and the j-invariant are closely related:

Proposition 2.5. *Two isomorphic elliptic curves E and E' defined over the same field K have the same j-invariants: $j(E) = j(E')$. Conversely, two curves having the same j-invariant are isomorphic over the algebraic closure $\overline{K}$ of K.*

Proof. See [Sil92, Proposition III.1.4]. □

In the other case E and E' are named twists of each other:

Definition 2.17 (Twist)**.** We are given two elliptic curves E and E' over the field K. If $j(E) = j(E')$, but $E(K) \not\simeq E'(K)$, then E and E' are *twists* of each other.

A quadratic twist of a curve is defined in the following manner:

Definition 2.18 (Quadratic twist)**.** Let E be an elliptic curve over some finite field K. Its *quadratic twist* E' is then:

- If $char(K) = 2$, E is of the shape $Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$ and if $d \in K$ has trace 1, then

$$E' : Y^2 + a_1XY + a_3Y = X^3 + (a_2 + da_1^2)X^2 + a_4X + a_6 + da_3^2.$$

- If $char(K) \notin \{2, 3\}$, E is of the shape $Y^2 = X^3 + a_4X + a_6$ and if d is not a square in K^*, then

$$E' : Y^2 = X^3 + d^2a_4X + d^3a_6.$$

The quadratic twist is isomorphic to the original curve over the algebraic closure $\overline{K}$ and is unique up to isomorphisms. In a similar way it is possible to specify cubic twists and the like.

Example 2.9. Let $E : Y^2 = X^3 + X + 2$ be an elliptic curve over $\mathbb{F}_5$. Since 3 is a non-square in $\mathbb{F}_5$, we set $d = 3$. Then, the (quadratic) twist of E is

$$E' : Y^2 = X^3 + 4X + 4.$$

Birational Equivalences

There are certain classes of elliptic curves, called birational equivalence classes. Within such a class any two curves are connected through two rational functions in the following way:

Definition 2.19 (Birational Equivalence)**.** Let K be any field. Two (affine) elliptic curves E/K and E'/K are called *birationally equivalent*, if there are two rational functions $\phi = (\phi_1, \phi_2)$ and $\psi = (\psi_1, \psi_2)$ with $\phi_i \in K(E)$, $\psi_i \in K(E')$ for $i = 1, 2$ such that:

- $\phi : E \mapsto E'$,
- $\psi : E' \mapsto E$,
- $\phi \circ \psi = id_{E'}$, and
- $\psi \circ \phi = id_E$.

This means that ϕ and ψ are inverse to each other and that we can use them to map points between the two curves E and E'. Such an birational equivalence is almost as powerful as a curve isomorphism. However, since the denominator of a rational function has finitely many roots, both ϕ and ψ are undefined for a finite number of points. Such points are said to be *exceptional*.

Curve Isogenies

Another notion we are going to use later, is the notion of an isogeny.

Definition 2.20 (Isogeny, isogeneous curves)**.** An *isogeny* over K is a group homomorphism ϕ between two elliptic curves E, E' over K sending the neutral element of E to the neutral element of E':

$$\phi : E \to E'$$

with $\phi(\mathcal{O}_E) = \mathcal{O}_{E'}$. Consequently, the two curves E, E' are considered to be *isogenous* over K if there exists a non-constant isogeny $\phi : E \to E'$ (that is $\phi(E) \neq \{\mathcal{O}_{E'}\}$) over K.

To every isogeny $\phi : E \to E'$ belongs an isogeny $\phi' : E' \to E$ called *dual isogeny*, such that $\phi \circ \phi'$ is the endomorphism of multiplication by $m \neq 0$. m is called the *degree of* ϕ, written $\deg \phi$. An isogeny of degree m is called *m-isogeny*.

The next theorem has an impact on isogenies in connection with elliptic curves defined over finite fields.

Theorem 2.2 (Tate)**.** *Two elliptic curves E, E' are isogenous over $\mathbb{F}_q$ if and only if $|E(\mathbb{F}_q)| = |E'(\mathbb{F}_q)|$.*

Proof. We refer to [Tat66]. □

2.4.5 Alternative Curve Coordinates

The default addition law is inappropriate for many applications. Not only that it is vulnerable to side-channel attacks but also because of its mediocre speed. The regular addition formula of two points on Weierstrass curves takes one inversion as well as several multiplications and squarings, where the field inversion is the most expensive part. It is estimated that an inversion lasts about 9 to 40 times longer than a single field multiplication [CF05, Section 13.2.1.d]. Switching to alternative curve coordinates may, to some extent, help avoiding such costly field operations and can hence be beneficial.

This subsection presents the most important types of coordinates aside from Edwards coordinates, which are the topic of Part III.

Projective Coordinates

The switch to projective coordinates allows us to omit the inconvenient inversion. The projectively closed curve in the non-binary case is

$$Y^2Z = X^3 + a_4XZ^2 + a_6Z^3.$$

In the binary case we restrict us to the curve equation

$$Y^2Z + XYZ = X^3 + a_2X^2Z + a_6Z^3.$$

The point at infinity is then $\mathcal{O} = (0 : 1 : 0)$ and the projective point $(x_1 : x_2 : x_3)$ corresponds to $(x_1/z, x_2/z)$ provided that $z \neq 0$.

Projective Addition and Doubling Formulae In order to derive projective addition formulae, we start with a conversion of the two addends P_1 and P_2 to affine points P_1' and P_2'. Next, P_1' and P_2' are plugged into the affine addition formula. Finally, the result P_3' is reconverted to a projective point P_3, where the z-coordinate P_3 is chosen in such a way that the denominators in the affine formula are being cleared. Analogously, it is possible to derive projective doubling formulae:

Example 2.10 (Point doubling formula for projective coordinates)**.** We derive the projective doubling formula for the projective curve

$$E : Y^2Z = X^3 + a_4XZ^2 + a_6Z^3.$$

The affine counterpart of E is

$$E' : Y^2 = X^3 + a_4X + a_6.$$

Let $P_1 = (x_1 : y_1 : z_1) \in E$ satisfying $P_1 \neq -P_1$. Then, $P_1' = (x_1/z_1, y_1/z_1) \in E'$. After plugging P_1' into the affine doubling formula we get $P_3' = (x_3', y_3')$ with:

$$x_3' = \lambda^2 - 2(x_1/z_1),$$

$$y_3' = \lambda(x_1/z_1 - x_3') - y_1/z_1,$$

where $\lambda = (3(x_1/z_1)^2 + a_4) \cdot (2y_1/z_1)^{-1}$.

After simplifying the above results we obtain:

$$x_3' = ((3x_1^2 + a_4z_1^2)^2 - 8X_1y_1^2z_1) \cdot (4y_1^2z_1^2)^{-1},$$

$$y_3' = ((3x_1^2 + a_4 z_1^2) \cdot (2y_1 z_1)^{-1})^2 (x_1/z_1 - x_3') - y_1/z_1.$$

Evidently, choosing $Z_3 = 4Y_1^2 Z_1^2$ clears the denominator when reconverting to projective coordinates:

$$x_3 = (3x_1^2 + a_4 z_1^2)^2 - 8x_1 y_1^2 z_1,$$
$$y_3 = (3x_1^2 + a_4 z_1^2)^2 (4x_1 y_1^2 z_1 - x_3) - 4y_1^3 z_1,$$
$$z_3 = 4y_1^2 z_1^2.$$

For a comprehensive listing of projective arithmetic formulae take a look at [CF05, Section 13.2.1.b].

Jacobian Coordinates

Jacobian coordinates are a derivative of projective coordinates. Compared to projective coordinates they offer amazingly fast doubling formulae at the expense of additional costs for addition. Instead of associating a projective point $(x : y : z)$ to an affine point $(x/z, y/z)$ we associate it to the point $(x/z^2, y/z^3)$. Consequently, the *Jacobian curve* has equation:

$$E : Y^2 = X^3 + a_4 X Z^4 + a_6 Z^6$$

in the non-binary case, and

$$E' : Y^2 + XYZ = X^3 + a_2 X^2 Z^2 + a_6 Z^6$$

in the binary case with non-zero j-invariant. The point at infinity corresponds to $(1 : 1 : 0)$ and the arithmetic formulae can be derived in the same fashion as for projective coordinates. For a comprehensive listing of Jacobian arithmetic formulae please have either a look at [CF05, Section 13.2.1.c] or at [HMV04, Algorithms 3.21 and 3.22].

Montgomery Curves

Montgomery-type elliptic curves were initially used in [Mon87] for special curves over fields of large characteristic. Generalizations to other curves and to curves over binary fields do exist [LD99].

An elliptic curve in *Montgomery form* over field $K, char(K) \notin \{2, 3\}$ is defined as:

$$E^{M,a,b} : bY^2 = X^3 + aX^2 + X$$

where $a, b \in K$ satisfying $a \neq \pm 2, b \neq 0$. The set of K-rational points of $E^{M,a,b}$ together with the point at infinity $\mathcal{O} = (0 : 1 : 0)$ forms the group $E^{M,a,b}(K)$. Every Montgomery-shape curve is isomorphic to a curve in short Weierstrass form, but not the other way round. The subsequent proposition lists the conditions under which a Weierstrass curve can be transformed to Montgomery form:

Proposition 2.6. *Fix an arbitrary Weierstrass curve $E : Y^2 = X^3 + a_4 X + a_6$ over K. Then, E is transformable to Montgomery shape if and only if*

- *the equation $X^3 + a_4 X + a_6 = 0$ has at least one solution $\alpha \in K$, and*
- *the element $3\alpha + a_4$ is a square in K.*

Proof. In [OKS00, Proposition 1] Katsuyuki Okeya, Hiroyuki Kurumatani and Kouichi Sakurai mention and prove this statement for $K = \mathbb{F}_p$. Yet, in the above setting the proof remains valid for arbitrary fields K with $char(K) > 3$. □

Please note that over finite fields of prime characteristic $\mathbb{F}_p$ every such curve's group order is divisible by 4. In contrast to the Weierstrass addition law the Montgomery addition law resists side-channel attacks and features efficient computations solely based on x-coordinates. For further information on the isomorphism, the group law et cetera we either refer to [CF05, Section 13.2.3] or to the research papers [OKS00, Duq04].

Chapter 3

Background on Hash Functions

Hash functions play an important role in contemporary cryptography. They are widely used to calculate "fingerprints" of messages, such that any change to a message, may it be intentional or by accident, can be revealed.

This chapter provides background knowledge on hash functions in general taken from [Pra07], on memoryless collision searches as discussed in [BB08, Saa04] and on a hashing algorithm called MuHASH [BM97]. All of those techniques are necessary for a better understanding of the following chapters.

3.1 Cryptographic Hash Functions

A *cryptographic hash function* H is an algorithm mapping a message of arbitrary length to a bitstring of fixed size m, called *hash value* or sometimes also called *message digest*. Famous examples for cryptographic hash functions are the algorithms MD5 [Riv92] and SHA-1 [EJ01].

In mathematical notation H is written as:

$$H : \{0,1\}^* \rightarrow \{0,1\}^m$$

For cryptographic purposes it is important that no two messages ever yield the same hash value. Hence, an *ideal hash function* satisfies four main features:

1. $H(M)$ is easy to compute for any given message M.
2. *Collision resistance:* it is computationally infeasible to find two distinct messages $M \neq M'$ with the same message digest, i.e. $H(M) = H(M')$.
3. *Preimage resistance:* for a given hash value h, it is computationally infeasible to compute a message M such that $H(M) = h$.
4. *Second preimage resistance:* for a given message M it is computationally infeasible to compute another message M' such that $H(M) = H(M')$.

Clearly, after evaluating more than 2^m different messages, there must be at least one collision. In fact, according to the birthday paradox we need not hash more than 2^m messages as we can expect a collision after trying about $\sqrt{2^m}$ messages:

Theorem 3.1 (Birthday Paradox)**.** *We are given an urn filled with n balls of different colors. In every step we draw one ball and put another ball of the same color back into the urn. Then, for larger n we can expect the first pair of coinciding balls after roughly* $\sqrt{\frac{n\pi}{2}} \approx 1.25\sqrt{n}$ *steps.*

Hence, a hash function is considered to be *broken*, if an exploit of its specific properties allows collisions to be found in asymptotically less than $\sqrt{2^m}$ steps or preimages and second preimages to be found in asymptotically less than 2^m steps, respectively.

Originally, the label "birthday paradox" comes from a special case of Theorem 3.1, which says that in a randomly chosen group of at least 23 people there are two people born on the same day with probability greater than $\frac{1}{2}$. Attacks based on the birthday problem are called *birthday attacks*. Birthday attacks are general attacks and as such they are applicable to every hash function. In order to mount such an attack, the message digests of the random messages are put in a hash table until a collision occurs. Hence, such an attack does not only take $O(\sqrt{2^m})$ computational steps, but also $O(\sqrt{2^m})$ memory. The next section provides a memory-sparing alternative to this approach.

3.2 Floyd's Cycle-Finding Algorithm

In hash function analysis cycle-finding algorithms are widely used to perform memoryless collision searches. One of the first such algorithms was Floyd's cycle-finding algorithm, also known as *"tortoise and hare"*-algorithm. It traces back to an algorithm proposed by Robert W. Floyd that lists cycles in directed graphs [Flo67]. In further consequence this algorithm was extended to detect collisions of hash functions. In 1969, Donald E. Knuth was the first, who gave a description of how cycles can be found in functional graphs using the *"tortoise and hare"*-algorithm [Knu69].

3.2.1 The Idea behind Floyd's Cycle-Finding Algorithm

Now, let us take a look at the theoretical background of this algorithm. At first, we consider a pseudorandom sequence generated followingly:

$$x_{i+1} = H(x_i),$$

where $H : \{0,1\}^* \to \{0,1\}^m$ is either a hash function or some other pseudorandom function returning bitstrings of length $m \in \mathbb{N}$. Since the range of H is limited, the sequence $(x_i)_{i\geq 0}$ is periodic and a collision must occur at some point. This means that there will be a pair (x_{i_1}, x_{i_2}), such that $x_{i_1} \neq x_{i_2}$ and $H(x_{i_1}) = H(x_{i_2})$. According to the birthday paradox we can expect a collision after approximately $\sqrt{2^m}$ steps.

Definition 3.1. Assume that i'_1 and i'_2 are the smallest indices, such that $x_{i'_1} \neq x_{i'_2}$ and $H(x_{i'_1}) = H(x_{i'_2})$. Then, $\mu = i'_1 + 1$ is called the *tail length* of the cycle and $\lambda = i'_2 - i_1$ is called its *cycle length*.

Obviously, $x_i = x_{i+c\lambda}$ holds for all $i \geq \mu$ and for all $c \in \mathbb{N}$. Thus, if index i attains the value $c\lambda$, i.e. a multiple of the cycle length, we have

$$x_i = x_{i+c\lambda} = x_{2i}.$$

From this it follows, that in order to determine such a multiple $c'\lambda$ of λ it suffices to look at pairs (x_i, x_{2i}) starting from (x_0, x_0).

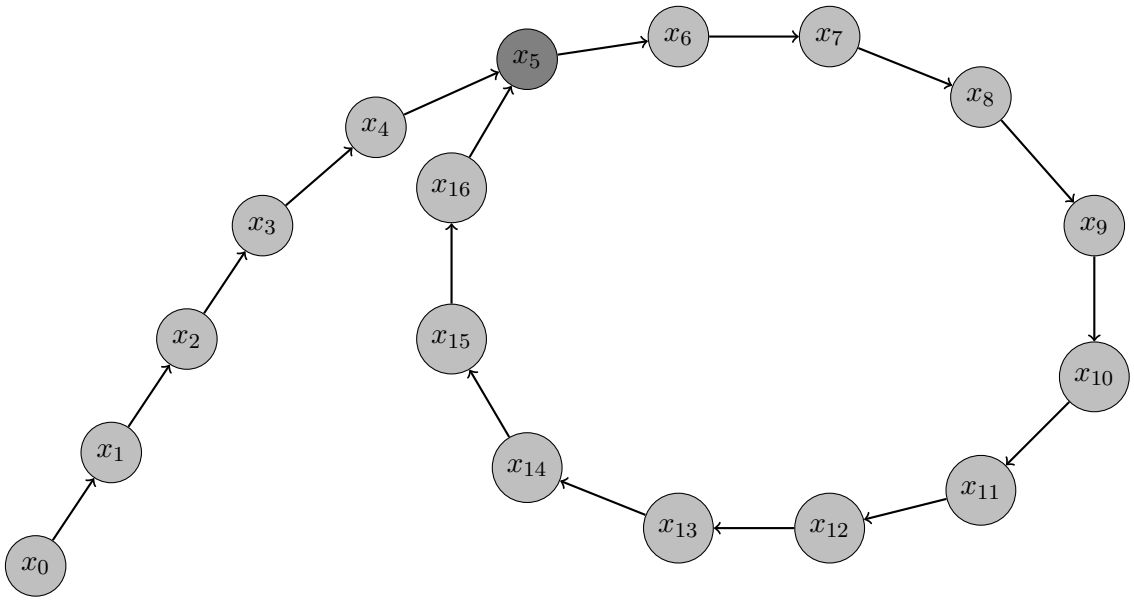

Figure 3.1: Illustration of Floyd's cycle finding algorithm ($\mu = 5, \lambda = 12$)

Theorem 3.2 (Knuth)**.** *For every periodic sequence $x_0, x_1, x_2, \ldots$ there is an index $i > 0$ such that $x_i = x_{2i}$ holds. The smallest such i lies between $\mu \leq i \leq \mu + \lambda$.*

Proof. The former statement follows from the previous discussion. The latter statement is obvious as, firstly, i must not be the index of a tail element (that is $i \geq \mu$) in order to yield a collision and, secondly, a multiple of λ lies between μ and $\mu + \lambda$. □

Next, we determine the sequence

$$(x_0, x_{c'\lambda}), (x_1, x_{c'\lambda+1}), \ldots, (x_\mu, x_{c'\lambda+\mu})$$

and stop as soon as $H(x_i) = H(x_{c'\lambda+i})$ to get the first collision $(x_{\mu-1}, x_{\mu+c'\lambda-1})$.

All in all, the first part of this algorithm takes $3c'\lambda$ invocations of H and $c'\lambda$ comparisons, whereas the second part takes 2μ hash function invocations and μ comparisons. Algorithm 3.1 details Floyd's cycle finding algorithm in pseudocode.

3.2.2 Expected Cycle and Tail Lengths

This subsection deduces the expected tail and cycle lengths μ and λ. We start with the probability distribution of $\lambda + \mu$. In the i^{th} step we can choose between all, except for the already outputted elements, if we want to avoid duplicates. This means that we can choose between $n - i + 1$ out of $n = 2^m$ elements. Therefore, the probability that $\lambda + \mu$ is greater than k is given by:

$$P(\lambda + \mu > k) = \frac{n^{\underline{k}}}{n^k}$$

Now, we are going to deduce the expected number of steps before the first collision occurs: $\mathbb{E}(\lambda + \mu)$. For this purpose the probability measure of $\lambda + \mu$ is rather unhandy. First of all, we derive a suitable approximation:

$$P(\lambda + \mu > k) = \frac{n}{n} \cdot \frac{n-1}{n} \cdot \ldots \cdot \frac{n-(k-1)}{n} = \left(1 - \frac{1}{n}\right) \cdot \left(1 - \frac{2}{n}\right) \cdot \ldots \cdot \left(1 - \frac{k-1}{n}\right)$$

Algorithm 3.1 Floyd's cycle finding algorithm

Require: $H : \{0,1\}^* \to \{0,1\}^m$ to be a pseudorandom function, for some $m \in \mathbb{N}$.
Ensure: $x \neq y$ and $H(x) = H(y)$

```
x_i = x_0
x_2i = x_0
repeat {Find a repetition x_i = x_2i}
    x_i = H(x_i) {Move with regular speed (tortoise step)}
    x_2i = H(H(x_2i)) {Move twice as fast (hare step)}
until x_i = x_2i
x_2i = x_i
x_i = x_0
repeat {Determine the position of the first repetition}
    x = x_i {Store x_i}
    y = x_2i {Store x_2i}
    x_i = H(x_i)
    x_2i = H(x_2i)
until x_i = x_2i
return (x, y)
```

After applying the logarithm we get:

$$\log(P(\lambda + \mu > k)) = \sum_{i=1}^{k-1} \log\left(1 - \frac{i}{n}\right) \tag{3.1}$$

The logarithm has the following well-known power series:

$$\log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots,$$

which applied to Equation (3.1) yields:

$$\log(P(\lambda + \mu > k)) = \sum_{i=1}^{k-1} \log\left(1 - \frac{i}{n}\right) =$$
$$= \sum_{i=1}^{k-1} \left(-\frac{i}{n} - \frac{i^2}{2n^2} - \frac{i^3}{3n^3} - \frac{i^4}{4n^4} - \ldots\right) =$$

When we assume $k = O\left(n^{\frac{1}{2}+\epsilon}\right)$, we have:

$$= \sum_{i=1}^{k-1} \left(-\frac{i}{n} + O\left(n^{-1+2\epsilon}\right)\right) =$$
$$= -\frac{1}{n}\sum_{i=1}^{k-1} i + O\left(n^{-\frac{1}{2}+3\epsilon}\right) =$$
$$= -\frac{1}{n}\frac{k(k-1)}{2} + O\left(n^{-\frac{1}{2}+3\epsilon}\right) =$$
$$= -\frac{k^2}{2n} + O\underbrace{\left(n^{-\frac{1}{2}+3\epsilon}\right)}_{\to 0 \text{ as } n\to\infty}$$

As result we get:

$$P(\lambda + \mu > k) \approx \exp\left(-\frac{k^2}{2n}\right)$$

for $k = O(n^{\frac{1}{2}+\epsilon})$. Now, we are able to calculate the mean value $\mathbb{E}(\lambda + \mu)$:

$$\mathbb{E}(\lambda + \mu) = \sum_{k\geq 0} k \cdot P(\lambda + \mu = k) = \sum_{k\geq 0} k \cdot (P(\lambda + \mu > k - 1) - P(\lambda + \mu > k)) =$$

which simplifies to

$$= \sum_{k\geq 0} P(\lambda + \mu > k) \approx \sum_{k\geq 0} e^{-\frac{k^2}{2n}}$$

In order to evaluate the limit of this series, we can approximate it using the Euler-Maclaurin formula:

$$\sum_{k\geq 0} f(k) \approx \lim_{l\to\infty} \left(\int_0^l f(k)dk + \frac{f(0) + f(l)}{2} + \sum_{j=1}^{l} \frac{B_{2j}}{(2j)!} \left(f^{(2j-1)}(l) - f^{(2j-1)}(0) \right) \right)$$

Note that B_{2j} stands for the *Bernoulli numbers*. After plugging in $f(k) = e^{-\frac{k^2}{2n}}$ we get:

$$\sum_{k\geq 0} e^{-\frac{k^2}{2n}} \approx \lim_{l\to\infty} \left(\int_0^l e^{-\frac{k^2}{2n}} dk + \frac{e^0 + e^{-\frac{l^2}{2n}}}{2} + \sum_{j=1}^{l} \frac{B_{2j}}{(2j)!} \left(f^{(2j-1)}(l) - f^{(2j-1)}(0) \right) \right) =$$

$$= \lim_{l\to\infty} \left(\int_0^l e^{-\frac{k^2}{2n}} dk + \frac{e^0 + e^{-\frac{l^2}{2n}}}{2} + \sum_{j=1}^{l} \frac{B_{2j}}{(2j)!} \left(-\frac{l}{n} \right)^{2j-1} e^{-\frac{l^2}{2n}} \right)$$

where $f^{(2j-1)}(l) = \left(-\frac{l}{n}\right)^{2j-1} e^{-\frac{l^2}{2n}}$.

It would be convenient, if only the integral remained. Fortunately, this is almost the case, as

$$\lim_{l\to\infty} \frac{e^0 + e^{-\frac{l^2}{2n}}}{2} = \frac{1}{2} \text{ and } \lim_{l\to\infty} f^{(2j-1)}(l) = 0.$$

Thus, the sum

$$\sum_{j=1}^{l} \frac{B_{2j}}{(2j)!} \left(-\frac{l}{n} \right)^{2j-1} e^{-\frac{l^2}{2n}}$$

vanishes and it is reasonable to approximate the mean value $\mathbb{E}(\lambda + \mu)$ by the integral $\int_0^\infty e^{-\frac{k^2}{2n}} dk$. Finally, we solve the aforementioned integral and obtain an explicit formula:

$$\mathbb{E}(\lambda + \mu) \approx \int_0^\infty e^{-\frac{k^2}{2n}} dk =$$

We substitute $x = \frac{k}{\sqrt{2n}}$ and $dk = \sqrt{2n}dx$ and get:

$$= \sqrt{2n} \int_0^\infty e^{-x^2} dx =$$

After substituting $t = x^2$ and $dx = \frac{1}{2\sqrt{t}}dt$ we have:

$$= \sqrt{\frac{n}{2}} \int_0^\infty t^{-\frac{1}{2}} e^{-t} dt = \sqrt{\frac{n}{2}} \cdot \Gamma\left(\frac{1}{2}\right) = \sqrt{\frac{n\pi}{2}}$$

Thus, the expected value

$$\boxed{\mathbb{E}(\lambda + \mu) \approx \sqrt{\frac{n\pi}{2}} \approx 1.25\sqrt{n}} \tag{3.2}$$

The expected cycle and tail lengths $\mathbb{E}(\lambda)$ and $\mathbb{E}(\mu)$ are both $\sqrt{\frac{n\pi}{8}}$ [Har60].

3.2.3 Concluding Remarks

Next to Floyd's cycle finding algorithm several alternative techniques do exist. Brent's algorithm [Bre80], for instance, requires only one hash computation per iteration in exchange for three comparisons. Though, its expected runtime of $1.98\sqrt{n}$ iterations is rather high compared to $1.25\sqrt{n}$ iterations of Floyd's cycle finding algorithm, it is 25-30% faster if the comparisons are cheap [BB08]. Other approaches sacrifice the memorylessness in order to reduce the time complexity. Parallelized cycle-finding algorithms, like parallel variants of Floyd's algorithm (cf. [vOW94]), as well as a sequential algorithm by Nivash fall into this category [Niv04]. Nivash's algorithm uses a probabilistic logarithmic amount of memory and is considered to be the fastest cycle-finding algorithm on single core computers [CF05].

Chapter 4

Elliptic Curves in Cryptography

Starting with a section on the historical background of public key cryptography and a comparison between RSA and ECC this chapter takes a look at the usage of elliptic curves in cryptography. It focusses on elliptic curves over finite fields and reviews the security properties of different curve types. Moreover, it points out of what one has to be aware of when selecting a specific curve and sketches some state-of-the-art attacks. Finally, it deals with point compression and decompression, a technique that is used to save memory.

This chapter is mostly based on introductory lecture notes [TF03, OL08, Bir09a], on an excellent essay by Neal Koblitz et al. [KKM08], a master thesis [Kop09] as well as on several standard works [Sma09, CF05, HPS08, HMV04, Was08].

4.1 Historical Background

From ancient history until the 1970s cryptography had been solely based on symmetric methods. This means that the involved parties had been required to share the same secret in order to decipher the encrypted message. Such cryptographic secrets had been exchanged via couriers or in direct meetings. These methods involved both a certain amount of insecurity and trust into third parties. In 1976, Whitfield Diffie and Martin E. Hellman revolutionized the cryptographic key exchange with the invention of the first method that allowed the establishment of a shared secret over an unprotected communication channel [DH76]. The Diffie-Hellman key exchange is based on the difficulty of solving the DLP in groups $\mathbb{Z}_p$ modulo a large prime number p and was the originator of all other forms of asymmetric cryptography.

Shortly afterwards, in 1977, Ronald L. Rivest, Adi Shamir and Leonard Adleman introduced the RSA cryptosystem [RSA78], the first public key cryptosystem in history. It is based on the difficulty of factoring large integers, called integer factorization problem (IFP), and unlike the Diffie-Hellman method it can be directly used for data encryption. One novelty introduced through RSA is the substitution of one secret cryptographic key with one public key and one private key. Messages can be encrypted using a person's public key, whereas the private key is kept secret and can then be used to decrypt the message. So, this approach does not involve the necessity of secure key exchanges.

In 1985, Neal Koblitz [Kob87] and Victor S. Miller [Mil85] both discovered independently that elliptic curves over finite fields can be used for public key cryptography. Based on the difficulty of the ECDLP, they published a Diffie-Hellman type protocol for elliptic curves. Both of these two groundbreaking works were inspired by a paper of Hendrik

Lenstra, who had been the first who had found an application of elliptic curves in cryptography. In 1984, Lenstra had introduced a new factoring method based on elliptic curves [Len87], called elliptic curve method (ECM).

For some years it seemed that, although still not in widespread use, ECC had become a powerful alternative to RSA. On the contrary, at the beginning of the 1990s, RSA became well established and it was virtually the only public key cryptosystem in use. At the same time new algorithms, such as the number field sieve, came up and forced the usage of stronger RSA keys, as it has lowered the complexity of factoring an n-bit RSA modulus from $O(e^{n^{1/2+\epsilon}})$ to $O(e^{n^{1/3+\epsilon}})$. In further consequence, the required RSA key sizes have grown to lengths of thousand bits and more, which have rendered the RSA cryptosystem almost unmanageable in constrained environments. Apparently, this would have been a perfect opportunity for a break-through of ECC.

Then, however, in 1991 the Weil pairing attack [MOV91] was discovered, which allows an efficient computation of the discrete logarithm on a certain type of curves, the so-called supersingular curves. This was a major impact since these curves had been used very often as they provide a fast addition law.

This discovery and the failure of the closely related hyperelliptic curve cryptography [Kob89, ADH94] shattered the trust in elliptic curve cryptography for almost a decade. Additionally, this distrust was stirred up by the RSA fraction, who claimed that elliptic curves were not ready for prime-time use and did not have the same well studied mathematical background that RSA had at that point of time.

At the end of the 1990s the trust in the security of elliptic curve cryptosystems could slowly be re-established as no more security flaws had been discovered and industrial standards bodies supported ECC. A reason for this was also NSA's support for ECC.

Over time more and more cryptographic algorithms, such as DSA and El-Gamal were adapted to work with elliptic curves and also new applications, such as pairing based cryptography, which is a form of identity based cryptography, were found for it [SOK00, Jou00].

In the past couple of years much effort has been put into improvements of ECC, mostly into advances of the curve arithmetic. In 2007, Harold M. Edwards discovered a normal form for elliptic curves [Edw07] over number fields, which was revisited by Bernstein and Lange [BL07b]. Edwards' normal form was groundbreaking as it features a complete and also fast addition law. Bernstein and Lange adapted Edwards' normal form to curves over finite fields and soon found more and more extensions and appliances of the so-called Edwards curves [BL07c, BLF08, BBJ$^+$08, BL07a, Ber09].

4.2 ECC versus RSA

In this section we are going to compare the RSA cryptosystem with ECC cryptosystems.

As we have already heard in the former section, the security of RSA is based on the difficulty of factoring large integers, called the integer factorization problem (IFP), whereas the security of ECC cryptosystems relies on the difficulty of the DLP in elliptic curve groups. For the IFP several subexponential algorithms are known. The most powerful methods for factoring large integers are number field sieves, where the best number field sieve algorithm allows an n-bit integer to be factored in $O(e^{n^{1/3+\epsilon}})$ time. Neither these methods nor Lenstra's ECM method with a running time of $O(e^{n^{1/2+\epsilon}})$ nor index calculus algorithms, which can be used to determine the discrete logarithm in finite cyclic groups,

seem to be applicable to the ECDLP. Thus, no subexponential time algorithm solving the ECDLP has been found so far and it is assumed that the ECDLP problem is much harder than the IFP. Therefore, ECC algorithms can use smaller key sizes compared to RSA. By way of comparison, a 160 bit elliptic curve key provides more or less the same security as an 1024 bit RSA key [LV99], which yields a tremendous improvement in speed and memory consumption. This circumstance makes ECC especially suitable for constrained environments, like cell phones and smart cards.

4.3 Elliptic Curves over Finite Fields

For cryptographic purposes we focus on elliptic curves over finite fields $\mathbb{F}_q$. It is quite obvious that an elliptic curve group over the finite field $\mathbb{F}_q$ is itself finite. For each x-coordinate there are at most two y-coordinates plus the point at infinity and thus the number of elements is bounded by $2|\mathbb{F}_q| + 1$.

4.3.1 Elliptic Curves over Binary Finite Fields

Elliptic curves over binary finite fields are given through the following affine Weierstrass equation:

$$\boxed{E : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 \text{ with } a_i \in \mathbb{F}_{2^n} \text{ and } \Delta(E) \neq 0} \quad (4.1)$$

In the binary case there are two different possible types of curves. Curves given by the equation:

1. $Y^2 + a_3Y = X^3 + a_4X + a_6$ are said to be *supersingular*, whereas
2. $Y^2 + XY = X^3 + a_2X^2 + a_6$ are called *non-supersingular*.

Note also that E defined over some field with characteristics 2 is supersingular if and only if its j-invariant $j(E) = 0$. Subsections 4.5.2 and 4.6.1 explain why supersingular curves should be avoided for safety's sake.

4.3.2 Elliptic Curves over Finite Fields with Characteristic > 3

Elliptic curves over fields $\mathbb{F}_q$ of characteristic p greater than 3 are defined by a slightly simpler affine equation:

$$\boxed{E : Y^2 = X^3 + a_4X + a_6 \text{ with } a_i \in \mathbb{F}_p \text{ and } \Delta(E) \neq 0} \quad (4.2)$$

Example 4.1. The curve $Y^2 = X^3 - X$ defined over $\mathbb{F}_7$ has the $\mathbb{F}_7$-rational points $(0,0), (1,0), (4,2), (4,5), (5,1), (5,6), (6,0)$ plus $\mathcal{O}$.

4.3.3 The Group Order of Elliptic Curves over Finite Fields

So far, all we know is that an elliptic curve consists of the points satisfying the affine curve equation plus the point at infinity $\mathcal{O}$. Counting the number of elements can be accomplished in polynomial time.

The equation

$$\boxed{|E(\mathbb{F}_q)| = q + 1 - t} \quad (4.3)$$

determines the cardinality of an elliptic curve group. t is called *trace of Frobenius* and is an important curve parameter. Hasse's theorem gives an upper bound for t:

Theorem 4.1 (Hasse)**.** $|t| \leq 2\sqrt{q}$.

Proof. See for instance [Was08, Theorem 4.1]. □

More exactly, the number of points on an elliptic curve $E(\mathbb{F}_p)$ with prime $p > 3$ and equation $y^2 = f(x)$ is:

$$|E(\mathbb{F}_p)| = \underbrace{1}_{\mathcal{O}} + \sum_{x \bmod p} \underbrace{\left(1 + \left(\frac{f(x)}{p}\right)\right)}_{\in\{0,1,2\}} = 1 + p + \sum_{x \bmod p} \left(\frac{f(x)}{p}\right), \tag{4.4}$$

where $\left(\frac{x}{p}\right)$ stands for the Legendre symbol. This is correct, because of point (x, y) being counted once, if $f(x) = 0$, twice, if $f(x)$ is a quadratic residue and never, if $f(x)$ is a quadratic non-residue. This result can be generalized to elliptic curves over arbitrary finite fields with characteristic greater than 3. In place of the Legendre symbol, we use its generalization:

$$|E(\mathbb{F}_q)| = 1 + \sum_{x \in \mathbb{F}_q} (1 + \chi(f(x))) = 1 + q + \sum_{x \in \mathbb{F}_q} (\chi(f(x)))$$

Note that this method requires exponential running time in the bitsize of p and q, respectively. In 1985, Rene Schoof published a deterministic polynomial-time algorithm for counting points on elliptic curves over finite fields with $p > 3$ [Sch85]. This is crucial, since the group order allows us to make a statement on the cryptographic properties of a specific curve. Section 4.4 gives more details on that.

For reasons of safety one usually operates on a prime-order subgroup of an elliptic curve generated by some generator G. The remaining factors of the group order $E(\mathbb{F}_q) = h \cdot ord(G)$ result in the so-called *cofactor* h.

Definition 4.1 (Cofactor)**.** Let $E/\mathbb{F}_q$ be an elliptic curve and G be the generator of a subgroup of $E(\mathbb{F}_q)$. Then

$$h = \frac{|E(\mathbb{F}_q)|}{ord(G)}$$

is an integer and is called the *cofactor*.

4.4 The Elliptic Curve Discrete Logarithm Problem

We have already covered the discrete logarithm and the DLP for groups in general in Section 2.1.1. In this section we give the definition of the *elliptic curve discrete logarithm* and the ECDLP.

Definition 4.2 (ECDL)**.** Let G be a generator (also called base point) and $P \in \langle G \rangle$. Then, the smallest, positive integer k: $P = k \cdot G = \sum_{i=1}^{k} G$ is called the *elliptic curve discrete logarithm* with respect to G.

This leads us directly to the definition of the *elliptic curve discrete logarithm problem*:

Definition 4.3 (ECDLP)**.** Given a point G of $(\mathbb{E}(\mathbb{F}_q), +)$ and a point $P \in \langle G \rangle$, the problem of finding an integer k such that:

$$P = k \cdot G$$

is called the *elliptic curve discrete logarithm problem (ECDLP).*

Finding such an integer in general is assumed to be hard, since no subexponential algorithm is known for this specific problem. Thus, the security of cryptographic elliptic curve algorithms usually relies on the difficulty of this problem.

4.4.1 Generic Attacks against the ECDLP

There are several generic attacks on the DLP that can be used to solve the ECDLP, but have nothing to do with the particular structure of elliptic curve groups. Generic algorithms were the first ones that were applied to the ECDLP.

The Pohlig-Hellman Algorithm

The Pohlig-Hellman algorithm was initially designed to solve the DLP in finite fields. It is the only known generic attack having implications on the choice of the curve parameters. It solves the DLP in prime-order subgroups and makes use of the Chinese remainder theorem to construct a solution for the original group. It is especially suitable, if the group order contains small factors.

Pollard's Rho Method

In 1975, John M. Pollard came up with a probabilistic algorithm for the DLP in generic groups [Pol75], which is based on the idea of Floyd's cycle finding algorithm (cf. Section 3.2). It is named after the ρ-pattern emerging in the sequence graph as depicted in Figure 3.1. In the same manner as Floyd's algorithm, Pollard's algorithm looks for collisions in pseudorandom sequences. Pollard's rho method fostered the development of cycle finding techniques and is still the best available attack against the DLP. Note that it can be applied to the IFP as well. In the following, we are going to discuss Pollard's rho method in its formulation for the ECDLP.

For a given a base point G and a point $H \in \langle G \rangle$, we want to determine the ECDL k of H so that $H = k \cdot G$. In order to do so, we set

$$P_i = a_i \cdot G + b_i \cdot H,$$

where $(a_i)_{i \geq 0}, (b_i)_{i \geq 0}$ are two integer sequences and the resulting sequence of points $(P_i)_{i \geq 0}$ is pseudorandom. Again, we can find a collision by looking at pairs (k_i, k_{2i}) starting from (k_0, k_0). If a collision $P_{i_1} = P_{i_2}, i_1 \neq i_2$ occurs, we have

$$a_{i_1} \cdot G + b_{i_1} \cdot H = a_{i_2} \cdot G + b_{i_2} \cdot H$$

and hence we have found k:

$$k = (a_{i_1} - a_{i_2}) \cdot (b_{i_2} - b_{i_1})^{-1}.$$

Obviously, also this algorithm has constant memory requirements and an expected runtime of $\sqrt{\frac{n\pi}{2}}$ steps.

4.5 Choice of Curve Parameters

Badly chosen curves can reduce the security of elliptic curve systems drastically, since wrong chosen curve parameters can make the discrete logarithm calculable in subexponential or even polynomial time. Therefore, it is crucial to be aware of the main pitfalls, before choosing a specific curve.

4.5.1 Resistance against Generic Attacks

As we have just seen, the Pohlig-Hellman algorithm makes use of small prime factors of the group order. Therefore, it is essential to choose a base point $G \in E(\mathbb{F}_q)$ with fairly large prime order n and $|E(\mathbb{F}_q)|$ to be prime or almost prime, that is, $|E(\mathbb{F}_q)| = hn$ with $1 \leq h \leq 4$. This is the reason why elliptic curve standards, like [Nat00], [Int05] or [Sta00], recommend curves with cofactor $h \leq 4$ or even with $h = 1$.

4.5.2 Weak Elliptic Curves

The trace of Frobenius t indicates whether a curve can be considered cryptographically weak or strong with respect to the curve's discrete logarithm problem. This brings us to the subsequent definition:

Definition 4.4. We are given a curve $E(\mathbb{F}_q)$ with $q = p^n$ and *trace of Frobenius* t.

- If $t = 1$, the curve is said to be *anomalous*. If additionally $q = p$ holds the curve is *prime-field-anomalous* and weak.
- When p divides t the curve is said to be *supersingular* and is weak as well.

Example 4.2. The curve $E : Y^2 = X^3 - X$ defined over $\mathbb{F}_p$ is supersingular if $p \equiv 3 \bmod 4$. As shown in Example 4.1, for $p = 7$ we have $|E(\mathbb{F}_7)| = 8 = 7 + 1 - t$. This implies that the trace of Frobenius t equals 0. Obviously, we have $7|0$ and so E is supersingular.

For safety's sake such curves should be avoided in cryptographic applications. The next section explains why.

4.6 State of the Art Attacks on Elliptic Curves

The first sub-exponential attack against the ECDLP, the MOV attack, was published in 1991. It is named after its inventors Alfred J. Menezes, Tatsuaki Okamoto and Scott Vanstone [MOV91] and is also known as Weil pairing attack. A generalization published by Gerhard Frey, Hans-Georg Rück et al. applies to curves $E/\mathbb{F}_{p^n}$ with group order $|E(\mathbb{F}_{p^n})|$ coprime to p [FR94, FMR99]. Consequently, it seems handy to use for instance curves with $|E(\mathbb{F}_q)| = q$ (i.e. curves with trace of Frobenius 1). However, as it turned out, the ECDLP on such curves is even weaker since the SSSA attack [Sem98, SA98, Sma99], which applies to $q = p$ and its generalization by Hans-Georg Rück [Rue99] to arbitrary finite fields, break it in polynomial time.

Section 4.6.1 deals with the MOV attack and Section 4.6.2 with the SSSA attack.

Algorithm 4.1 The MOV algorithm

Require: $P, Q \in E[l]$, $ord(P) = l$ and $l \in \mathbb{P}_{\geq 3}$
Ensure: $Q = b \cdot P$
1: **repeat**
2: Pick a random point $T \in E(\mathbb{F}_{p^k})$.
3: Compute $\alpha = e_l(P, T)$.
4: **until** $\alpha \neq 1$ {Ensure that P and T are independent}
5: Compute $\beta = e_l(Q, T)$.
6: Use a subexponential algorithm to find $b \in \mathbb{N}$ such that $\beta = \alpha^b$. Then also $Q = b \cdot P$.
7: **return** b.

4.6.1 The Weil Pairing Attack

Supersingular curves can be attacked using the Weil pairing attack [MOV91]. This attack was the first use of the Weil pairing in cryptography. It transfers the DLP of a supersingular curve $E(\mathbb{F}_q)$ to the DLP of an extension field of $\mathbb{F}_q$, where we can solve the latter one in subexponential time using for instance index calculus.

The Weil Pairing

The Weil pairing is an efficiently computable non-degenerate bilinear mapping:

$$e_m : E[m] \times E[m] \rightarrow \mu_m(\mathbb{F}_{q^k}),$$

that is:

1. $e_m(P + Q, R) = e(P, R) \cdot e(Q, R)$
2. $e_m(P, Q + R) = e(P, Q) \cdot e(P, R)$
3. If $e_m(P, Q) = 1$ for all $Q \in E[m]$ then $P = \mathcal{O}$ *(non-degenerate)*

Furthermore, the Weil pairing has some additional features:

1. $e_m(P, Q)^m = 1$ for all $P, Q \in E[m]$, which implies that $e_m(P, Q)$ is an m^{th} root of unity.
2. It is *alternating*, i.e. $e_m(P, P) = 1$ for all $P \in E[m]$.

Note that k is called the curve's *embedding degree*. It is the smallest positive integer such that firstly m divides $(q^k - 1)$ and secondly, $\mathbb{F}_{q^k}$ is the smallest field comprising the coordinates of all points in $E[m]$.

Rationale

In order to mount the attack we assume a point P of prime order $l \geq 3$ and the Weil pairing $e_l : E[l] \times E[l] \rightarrow \mu_l(\mathbb{F}_{q^k})$. Now, we are able to compute the discrete logarithm of Q with respect to P as shown in Algorithm 4.1.

In the first four steps the algorithm chooses a point $T \in E[l]$, which is independent from P, i.e. neither T is a multiple of P nor vice versa. This is crucial, since e_m is alternating and otherwise we would get:

$$e_l(P, T) \overset{T=a \cdot P}{=} e_l(P, a \cdot P) \overset{\text{bilinear}}{=} e_l(P, P)^a = 1^a = 1$$

After computing the Weil pairing of Q and T in step 5, it uses a subexponential algorithm such as index calculus to solve the DLP in the finite field $\mathbb{F}_{q^k}$.

Why does this attack work? We know that $\beta = \alpha^b$, that $\beta = e_l(Q,T)$ and that Q is a multiple of point P, let us say $Q = a \cdot P$. What we have next is:

$$\beta = e_l(Q,T) = e_l(a \cdot P, T) \overset{\text{bilinear}}{=} e(P,T)^a = \alpha^a$$

Now, we have obtained two different representations of β:

$$\beta = \alpha^b \text{ and } \beta = \alpha^a.$$

From this it follows that:

$$\alpha^{(b-a)} = 1.$$

In other words $(b-a)$ is a multiple of the order l of $\mu_l(\mathbb{F}_{q^k})$', that is, $l|(b-a)$. Since the group order of $E[l]$ is the same, we get:

$$(b-a) \cdot P = \mathcal{O},$$

which implies

$$b \cdot P = a \cdot P = Q.$$

It is still to be clarified, why l has to be prime. The proposition below gives an answer to that:

Proposition 4.1. *Let $l \in \mathbb{P}_{\geq 3}$, E be an elliptic curve, $P, T \in E[l]$ and consider $E[l] = \mathbb{Z}_l \times \mathbb{Z}_l$ as vector space over the field $\mathbb{Z}_l$. Then the following statements are equivalent:*

- *P and T are a basis of $E[l]$.*
- *$P \neq \mathcal{O}$ and T are independent of each other.*
- *$e_l(P,T) \in \mu_l(\mathbb{F}_{q^k})$ is a primitive l^{th} root of unity.*
- *$e_l(P,T) \neq 1$.*

Proof. We refer to [HPS08, Proposition 5.49]. □

This proposition ensures that $\alpha = e_l(P,T)$ is a generator of $\mu_l(\mathbb{F}_{q^k})$ if and only if P and T are independent of each other. Hence, for all $\beta \in \mu_l(\mathbb{F}_{q^k})$ there is an $b \in \mathbb{N}$ such that $\beta = \alpha^b$.

The Role of the Embedding Degree

Before the Weil pairing attack was discovered in 1991 supersingular curves had been widely used as they are easy to compute with. This had the effect that for some time it was assumed that k should be fairly large, e.g. $k \geq 20$ or even $k \geq (q-1)/100$. But as it has turned out, this reaction was overdrawn. It seems that solving the DLP with current means is still computationally infeasible, as long as $k \geq 6$.

Nevertheless, the attack has had a major impact on the usage of supersingular curves, since this curve type has embedding degree $k \leq 6$. Except for some special appliances,

Algorithm 4.2 The SSSA algorithm

Require: $G, P \in E(\mathbb{F}_p)$ and $\phi(P) \neq 0$
Ensure: $P = l \cdot G$
1: Compute $a = \phi(P)$ and $b = \phi(G)$.
2: Find $l \in [0, p-1]$ such that $l \equiv ba^{-1} \bmod p$ using the extended Euclidean algorithm. Then $la \equiv b \bmod p$ and also $P = l \cdot G$.
3: **return** l.

like identity based cryptography, the use of supersingular curves is taboo. Fortunately, curves with small embedding degree are rare. A randomly chosen non-supersingular curve over $\mathbb{F}_p$ has probability $O(p^{-1})$ of having delimited embedding degree (see [BK98]) and the probability of selecting a supersingular curve at random is at most $O(p^{-1/2})$. Also note that the MOV attack is only reasonably fast, if the curve's embedding degree is small (see [Mil04]), namely $k \leq (\ln p)^2$. For larger embedding degrees the DLP in the finite field $\mathbb{F}_{q^k}$ is in fact harder to solve than the ECDLP on the curve.

4.6.2 The SSSA Attack

The SSSA attack, as illustrated in Algorithm 4.2, determines the discrete logarithm of some point P with respect to base point G by reducing the ECDLP of a prime-field-anomalous curve $E(\mathbb{F}_p)$ to the DLP of the additive group $(\mathbb{Z}_p, +)$ of integers modulo p. In this group the DLP can be solved easily using the extended Euclidean algorithm. Note that it only works for curves with prime group order p, because such curves are cyclic and in further consequence they are isomorphic to $(\mathbb{Z}_p, +)$. In 1997 such an isomorphism $\phi : E(\mathbb{F}_p) \to \mathbb{F}_p$ was proposed independently by Smart [Sma99], Semaev [Sem98] as well as by Satoh and Araki [SA98]. Since ϕ is efficiently computable, the whole attack is feasible.

4.7 Saving Space through Point Compression and Decompression

Point compression reduces the space required to store a point but still keeps the same amount of information. This method is essential in smart card As we know already, there are at most two points with the same x-coordinate on each elliptic curve. So, for some point P it suffices to use a single additional bit to determine whether $P = P_0$ or $P = -P_0$.

4.7.1 Point Compression and Decompression on Curves over Finite Fields of Characteristics Greater than 3

For an elliptic curve $E : Y^2 = X^3 + a_4X + a_6$ over $\mathbb{F}_q$ of odd characteristics $p > 3$, we have $P_0 = (x_0, y_0)$ and $-P_0 = (x_0, -y_0)$. They are equal if and only if $y_0 = 0$.

Compression

We save x_0 as well as the least significant bit of coefficient c_0 of the polynomial representation $y_0 = \sum_{i=0}^{n-1} c_i\alpha^i$ of y_0, which we denote by $b(y_0)$. Since the least significant coefficient of $-y_0$ is $p - c_0$ and p is an odd prime, the parity of $b(-y_0)$ is inverse to the parity of $b(y_0)$. That is why this approach works well.

Decompression

In order to recover the y-coordinate of P_0 from $(x_0, b(y_0))$, we calculate $\beta = x_0^3 + a_4 x_0 + a_6$. According to the curve equation $Y^2 = X^3 + a_4 X + a_6$ this element is a square in $\mathbb{F}_q$ and yields y_0^2. Now, taking the square root we obtain $(+y_0, -y_0)$ and recover the appropriate y-coordinate using $b(y_0)$.

4.7.2 Point Compression and Decompression on Non-Supersingular Curves over Binary Finite Fields

We map a point $P_0 = (x_0, y_0)$ of some curve $E : Y^2 + XY = X^3 + a_2 X^2 + a_6$ over $\mathbb{F}_{2^n}$ to its non-redundant shape $(x_0, b(y_0))$, where $b(y_0)$ is the bit distinguishing P_0 from $-P_0 = (x_0, x_0 + y_0)$.

Just like [CF05, Section 13.3.7], we explain decompression first as this task is easier.

Decompression

We are given the compressed coordinates $(x_0, b(y_0))$ and plug x_0 into the curve equation. As result we get the quadratic equation $Y^2 + x_0 Y = x_0^3 + a_2 x_0^2 + a_6$.

Apparently, if $x_0 = 0$, we have $y_0 = \sqrt{a_6} = a_6^{2^{n-1}}$. Otherwise, it has two solutions because of $x_0 \neq 0$ being a valid x-coordinate. To calculate these, we compute the element $\beta = (x_0^3 + a_2 x_0^2 + a_6) \cdot x_0^{-2} \in \mathbb{F}_{2^n}$ and solve the equation $Y^2 + Y = \beta$ for y_0'. This yields two roots, namely y_0' and $y_0' + 1$. If the rightmost bit of y_0' equals $b(y_0)$, we take the solution y_0' else we take $y_0' + 1$. The solution of the original equation is then: $y_0 = y_0' \cdot x_0$.

Point Testing With regard to elliptic curve hash functions (cf. Chapter 5) we would like to find valid points on curve E not only for compressed x-coordinates x_0, but for any $x_0 \in \mathbb{F}_{2^n}$. In order to determine whether some $P \in E(\mathbb{F}_{2^n})$ can be recovered from x_0, we test whether $Tr\left((x_0^3 + a_2 x_0^2 + a_6) \cdot x_0^{-2}\right) = 0$. If so, there is a $P \in E(\mathbb{F}_{2^n})$ having x-coordinate x_0. If not, we still have the possibility to change the value of x_0 in a prescribed way and then test it again.

Compression

From the description above emerges that $b(y_0)$ is equal to the least significant bit of $y_0' = y_0 \cdot x_0^{-1}$.

Part II

Hash Functions Based on Elliptic Curves

Chapter 5

The Elliptic Curve Only Hash (ECOH)

This chapter is dedicated to the so-called *Elliptic Curve Only Hash (ECOH)* [BACS08, Bro08b]. ECOH was proposed by Daniel R. L. Brown et al. in 2008 and participated in NIST's SHA-3 contest. In April 2009, Niels Ferguson and Michael A. Halcrow were able to find a second preimage attack against ECOH, which led to ECOH's knock-out in the first round of the SHA-3 competition [HF09]. Responding to this attack, Brown published a hardened version called ECOH^2 [Bro09].

This chapter starts with a comprehensive description of ECOH, points out its relation to MuHASH and describes some fresh attacks against ECOH. Then, we proceed to the aforementioned attack of Ferguson and Halcrow, detail our memoryless variant of it and show two colliding preimages computed using ECOH on a smaller curve. At last, we are going to discuss ECOH's improved version ECOH^2.

5.1 ECOH in Detail

ECOH is available in four different versions, namely ECOH-224, ECOH-256, ECOH-384 and ECOH-512. All those versions are based on the same generic algorithm, but differ in their parameters. The parameters are the following:

n hash output length in bits,

E NIST-recommended elliptic curve (cf. [Nat00, Section D.1.3]),

G base point of E,

m bitsize of underlying field $\mathbb{F}_{2^m}$,

$blen$ block bitlength,

$ilen$ index bitlength, and

$clen$ counter bitlength.

Table 5.1 gives a detailed listing of ECOH's parameter values.

Algorithm	n	E and G	m	$blen$	$ilen$	$clen$
ECOH-224	224	B-283	283	128	64	64
ECOH-256	256	B-283	283	128	64	64
ECOH-384	384	B-409	409	192	64	64
ECOH-512	512	B-571	571	256	128	128

Table 5.1: The four different versions of ECOH [BACS08, Table 4]

Algorithm 5.1 Generic ECOH pseudocode [BACS08, Table 5]

Require: Message M of maximum bitlength $len(M) < 2^{ilen}$.

1: Set $N = M \parallel 1 \parallel 0^j$ with j chosen minimally, such that $blen | len(N)$.
2: Split N into k blocks of bitlength $blen$: $N_0, \ldots, N_{k-1}$.
3: **for** $i = 0$ to $i = k - 1$ **do**
4: Index block N_i: $O_i = N_i \parallel I_i$, where I_i is the bit-representation of integer i of length $ilen$ bits.
5: **end for**
6: Compute the checksum block $O_k = \left(\bigoplus_{i=0}^{k-1} N_i\right) \parallel I_{len(M)}$, where $I_{len(M)}$ stands for the $ilen$-bit-representation of the message bitlength $len(M)$.
7: **for** $i = 0$ to $i = k$ **do**
8: Find bit string $X_i = \left(0^{m-(blen+ilen+clen)} \parallel O_i \parallel 0^{clen}\right) \oplus C_i$, where C_i is of length m and chosen minimally such that X_i belongs to a valid x-coordinate x_i of an element of the elliptic curve group $\langle G \rangle$.
9: Decompress point $P = (x_i, y_i)$ such that the leftmost bit of N_i equals the rightmost bit of $y_i \cdot x_i^{-1}$.
10: **end for**
11: Let $Q = \sum_{i=0}^{k} P_i$.
12: **return** n-bit representation of $\lfloor x_Q + \lfloor x_Q/2 \rfloor G)/2 \rfloor \bmod 2^n$.

5.1.1 The Algorithm in Detail

Algorithm 5.1 shows a generic version of ECOH's pseudocode. At first, ECOH pads an arbitrary input message M and obtains a new message N, which is then split up into parts $N_1, \ldots, N_{k-1}$, each of bitlength $blen$. In the third to fifth step it indexes each block and acquires new blocks O_i. Next, it calculates a checksum block containing the exclusive-or checksum of all blocks $N_1, \ldots, N_{k-1}$ and the length of M. Then, it tries to create valid x-coordinates for each block O_i and decompresses each x-coordinate to get a point on the curve. Note that this step is non-deterministic in the sense that C_i is being incremented until a valid x-coordinate is obtained. Finally, those points are added up to get a prehash value Q, which is then converted to an n-bit hash value in the last step.

5.1.2 Implementations of ECOH

The ECOH authors have submitted two different implementations of ECOH: a reference implementation and an optimized implementation. According to them [Bro08a], ECOH's reference implementation has a throughput of about 0.14 MB/s on a reasonably recent desktop and is as such almost 1000 times slower than SHA-1. However, they also emphasize that the optimized implementation is about three times faster and suggest various speedup

Algorithm	Output
ECOH-224	A92F0BA8488EBD27FCE100018DEF5373DE9FB45CCDE13DD56EEF69F1
ECOH-256	4E3C9B90A92F0BA8488EBD27FCE100018DEF5373DE9FB45CCDE13DD56EEF69F1
ECOH-384	D2B6B236D5EB8FA256C6F7B819830F22E99D3A61028854235F968F041247CBD2 ‖ 51FFBC3DF438C80A949ADF3C28686B76
ECOH-512	D537139E4061CB788993EAF801B108C7715C6EB34745332914B3069C975AA6D1 ‖ 6D33FDE39F7C9D1ED7D40B27924F29059CE7DE6FA7F8BCF5120D741AA293863F

Table 5.2: Output of ECOH on input 3EF6C36F20

techniques, such as simultaneous inversion, multicore support and the like, which may lead to a tremendous gain in speed. Finally, they also note that ECOH is incremental, which leads to a speedup in case one hashes several similar messages at once.

Example 5.1. Table 5.2 shows the message digests of ECOH's different versions on input 3EF6C36F20.

An Error in ECOH's Reference Implementation

We came across a grievous error in ECOH's reference implementation. As it turned out, ECOH's point decompression routine used to calculate

$$Tr(a_6 \cdot x^{-2})$$

instead of

$$Tr\left((x^3 + a_2x^2 + a_6) \cdot x^{-2}\right)$$

in order to determine whether x corresponds to a valid x-coordinate. Hence, ECOH's reference implementation hardly ever returned valid message digests.

5.2 MuHASH

This section gives an overview of MuHASH, which is a multiplicative hash function that served as a model for ECOH. MuHASH was proposed by Mihir Bellare and Daniele Micciancio in 1997 [BM97]. It is not a hash algorithm built from scratch but uses another collision-free hash function h to map data blocks to elements of some multiplicative group G where the DLP is hard. Those elements are then multiplied to obtain the hash result.

5.2.1 Properties

In its general construction, MuHASH is given by:

$$H^h_{\langle G \rangle}(M) = \prod_{i=1}^{blen(M)} h(i \parallel M_i)$$

where $i \parallel M_i$ is the i^{th} message block of M prepended with index i and $blen(M)$ is M's length in blocks. Typically, $G = \mathbb{Z}_p^*$ with p being at least a 1024 bit prime and multiplication modulo p. The size of the resulting hash is bounded by the bitlength of p.

Owing to G's associativity MuHASH is highly parallelizable and if G is chosen to be commutative, H becomes an *incremental hash function*. Suppose we have already computed the message digest of some message M and want to compute the digest of message M', which is the same as M except for one block M_j that has changed. Instead of reapplying MuHASH to the whole M', incrementality allows us to update $H(M)$ efficiently:

$$H(M') = H(M) \cdot h(j \parallel M_j)^{-1} \cdot h(j \parallel M'_j)$$

5.2.2 Security

What is so special about MuHASH is that its security is provable. Bellare and Micciancio were able to show that "as long as the discrete logarithm problem in G is hard and h is ideal, MuHASH is collision-free" [BM97, Section 1.3]. This means that if there is a way of finding collisions in MuHASH, then there must also be a way to solve the DLP of G efficiently. So, there is no way to attack MuHASH as long as the DLP of G is hard. Thus, it is not surprising that up to now no attack against MuHASH has been known. The closest attack on MuHASH is an attack by Wagner [Wag02] against a similar algorithm called AdHASH. AdHASH was introduced in one go with MuHASH but is much simpler and also faster than MuHASH (see [BM97, Section 5]).

5.2.3 ECOH's Relation to MuHASH

ECOH is based on the main ideas behind MuHASH and in this light it can be considered as a porting of MuHASH to elliptic curve groups. Just like MuHASH, ECOH

- is incremental,
- is parallelizable,
- uses an algebraic operation to calculate the message (pre-)hash, and
- operates on a group with a hard DLP.

There is, however, one big difference between ECOH and MuHASH. Namely, for practical reasons ECOH applies a padding function in place of an ideal hash function, which turned out to be its weak spot. MuHASH is proven to be collision-resistant under the assumption that the hash function, used by MuHASH to scramble the indexed data blocks, is ideal (cf. [BM97, Section 1.3]). This requirement, however, can not be fulfilled by ECOH's padding function, which does not hash but only extend the data blocks. Section 5.4 shows in which way this circumstance can be exploited.

5.3 New Attacks against Simplified Versions of ECOH

This section comprises algebraic and side-channel attacks on ECOH. Most of the algebraic attacks are new to our knowledge and exploit generic algebraic properties, such as torsion elements and the discrete logarithm, which becomes computable with the arrival of the quantum computer.

Algorithm 5.2 Computing torsion points

Require: $\langle G \rangle$ to be an elliptic curve group, $N = ord(G) > 1$ and N to be composite
Ensure: $ord(P) = l$
1: Factor N and obtain a factor $l > 1$.
2: Compute $\frac{N}{l} \cdot G = P \in E[l]$.
3: **return** P.

5.3.1 Torsion Element Attacks

This subsection contains attacks on ECOH's prehash. All of these attacks require a modified variant of ECOH that provides the points in the prehash sum. Modifying the algorithm in such a way is, however, an easy task. Note that all of these results remain applicable to the modified version ECOH^2 [Bro09]. These attacks will be successful, if there is a way to reconstruct preimage blocks of the points in the prehash sum, for the simple reason that all of these attacks solve equations of the type

$$P_0 + \ldots + P_k \pm \left((-P'_0) + \ldots + (-P'_{k'})\right) = \mathcal{O}.$$

Also note that if the previous requirements can be met, these attacks will allow an adversary to create collisions and second preimages of arbitrary messages.

How to Compute a Torsion Element

Algorithm 5.2 shows how to compute torsion elements of the elliptic curve group. It factors the group order N and computes a point of order l, for some l dividing N.

As for ECOH, N is sized between 283 and 571 bits. Hence, N can be factored in $O(e^{\log_2(N)^{1/3+\epsilon}})$ time using the number field sieve. This task becomes more difficult when we want to determine torsion elements of the elliptic curve group used by ECOH^2. In context of ECOH^2, the bitlength of N is 4 times larger than in ECOH, i.e. between 1132 and 2284 bits.

After N has been factored, the second step of Algorithm 5.2 can be achieved in $O(\log \frac{N}{l})$ time using for instance the square-and-multiply method (see [Knu69]).

Prehash Collision Attack

For all $l > 1$ with $l|N$, for every $P \in E[l]$ and for every non-symmetric partition $\{l_1, \ldots, l_k\}$ of l, i.e. $\sum_{i=1}^{k} l_i = l$, we can find two different ECOH prehashes, if we cut out the checksum block and the block indices:

$$\sum_{i=1}^{k_1} l_i \cdot P = \sum_{i=k_1+1}^{k} l_i \cdot (-P),$$

for all $1 \leq k_1 < k$. This works, because of

$$\sum_{i=1}^{k_1} l_i \cdot P + \sum_{i=k_1+1}^{k} l_i \cdot P = \sum_{i=1}^{k} l_i \cdot P = l \cdot P = \mathcal{O}$$

Another Prehash Collision Attack

The following attack will work, when we leave out the padding, the block indices as well as the message length in the checksum block.

According to [HMV04, Theorem 3.18(iii)], $|E(\mathbb{F}_{2^m})|$ is always even and divisible by 4 if the curve parameter a has trace 0. Thus, with the right choice of parameters, torsion points of order 2 do exist. Let M be an arbitrary message with $|M| \leq blen$, then its prehash $Q = P_M + C_M$, where C_M stands for the checksum point. Since we left out the indexing and the message consists only of one block, we get $C_M = P_M$, i.e. $Q = P_M + P_M$. Now, we assume one torsion point $P \in E[2]$ and get:

$$Q = P_M + P_M = \underbrace{(P_M + P)}_{P'} + \underbrace{(P_M + P)}_{\text{new checksum block}},$$

as $2P = \mathcal{O}$. If there is a way to recover the padding for new points, then the padding can also be incorporated.

An Idea for Obtaining Another Preimage: When we choose M, P such that $x(P_M + P)$ is either small or the lowest $clen$ bits of $x(P_M + P)$ and $x(P_M + P)$'s highest $m - (blen + ilen + clen)$ bits are 0, then the variable C_i in step 5 will very likely also be chosen to be small and we obtain another preimage M'.

A Second Preimage Attack

The following attack will work, when we leave out the padding, the block indices as well as the checksum block.

We are given the prehash $Q_M = P_1 + \ldots + P_k$ of some message M, where P_i corresponds to message block i. We can find a second prehash $Q_{M'}$ equal to Q_M by adding torsion points $P \in E[l]$ in the following manner, for instance:

$$Q_M = P_1 + \ldots + P_k = (P_1 + l'_1 P) + \ldots + (P_k + l'_k P) + \underbrace{l''_1 P + \ldots + l''_j P}_{\text{new blocks}} = Q_{M'},$$

where $0 \leq l' \leq l$, $\{l'_1, \ldots, l'_k\}$ is a partition of l' and $\{l''_1, \ldots, l''_j\}$ is a partition of $l - l'$.

5.3.2 A Quantum Attack against ECOH's Prehash

A quantum system can be in a so-called superposition of states meaning that all states possible can exist simultaneously. As for quantum computers this has the consequence that they are able to compute all solutions of a specific problem in one go. Hence, they operate non-deterministically, leading to an exponential speedup for many computational problems. And so, it is not rather surprising that many cryptographic problems, which are computationally infeasible for conventional deterministic computers, become solvable in polynomial time on quantum computers. This is the case for both the IFP and the DLP. In 1997, Peter W. Shor published polynomial-time quantum algorithms for both problems [Sho97], where his algorithm for the IFP has become famous as Shor's algorithm.

Thus, with a quantum computer an adversary might be able to calculate the discrete logarithm k of the prehash value Q of some input message M, i.e. $Q = k \cdot G$. Just like in the previous subsections, it is possible to obtain a second preimage, if we are able to

derive valid preimage blocks from G. In this way it may be possible to obtain another preimage M'.

5.3.3 A Timing Attack against ECOH's Point Search

A timing attack utilizes the side-channel data, which is being exhibited through the timing behavior of cryptographic devices, to derive information about the input or the cryptographic key in use.

We are given an input message M consisting of k blocks, which is being padded and split up into the $k+1$ blocks $N_0 \parallel \ldots \parallel N_k$. Now, we suppose that an adversary is able to mount a timing attack on an ECOH device. Then, in iteration i of the loop in line 7 of Algorithm 5.1, the adversary can observe how many times the counter C_i is being increased. In order to obtain a distinct second preimage block, the counter value C_i must affect the value of N_i embedded in X_i. Hence, we require $l \coloneqq len(C_i) - (clen + ilen) > 0$. If we can observe a value C_i satisfying this property, there is a good chance to receive a second preimage block N_i' by modifying the l least significant bits of block N_i, so that, if interpreted as integers, $N_i < N_i' \leq N_i + C_i$.

This approach is valid, since the point search routine returns the same point for the value of N_i as for the $C_i + 1$ subsequent block values. Therefore, we have found a second preimage $M = N_1 \parallel \ldots \parallel N_{i-1} \parallel N_i' \parallel N_{i+1} \parallel \ldots \parallel N_k$.

5.4 The Ferguson-Halcrow Second Preimage Attack

The Ferguson-Halcrow attack tries to find a second preimage to an arbitrarily fixed prehash value Q'. As ECOH's padding function is weak compared to MuHASH's ideal hash function, this attack is able to exploit this specific difference. To be more specific, ECOH's padding function opens up the possibility to generate certain sequences of message blocks having the same checksum and padding block.

For reasons of convenience the Ferguson-Halcrow attack works only for messages M with $blen|len(M)$, i.e. for messages with bitlength divisible by $blen$. Given k message blocks $M_0, \ldots, M_{k-1}$, we have the subsequent simplified characterization of ECOH:

$$\begin{aligned} X_1 &\coloneqq P'(n) \\ X_2 &\coloneqq P\left(\bigoplus_{i=0}^{k-1} M_i, n\right) \\ Q &\coloneqq \sum_{i=0}^{k-1} P(M_i, i) + X_1 + X_2, \end{aligned}$$

where Q is the resulting prehash point, $P(\cdot,\cdot)$ is the padding function that maps a message block and an indexing value to a point on E and $P'(\cdot)$ is the padding function computing the padding point X_1 whose value depends solely on the message length.

This attack operates on two lists L_1 and L_2, each of size K. The first comprises points of the shape $P(M_0, 0) + P(M_1, 1) + P(M_2, 2)$, whereas the second consists of points of the form $Q' - (P(M_3, 3) + P(M_4, 4) + P(M_5, 5) + X_1 + X_2)$. The messages M_2 and M_5 are both made up of the two preceding messages: $M_2 = M_0 \oplus M_1$ and $M_5 = M_3 \oplus M_4$. This has the advantage that the exclusive-or checksum of the message blocks $M_0, M_1, \ldots, M_5$ yields 0 and that in further consequence the checksum point X_2 does no longer depend on

the message blocks and remains constant. Also note that X_1 has been fixed too, since the message length equals $6 \cdot blen$ bits. If there is a match between L_1 and L_2 we obtain

$$\sum_{i=0}^{2} P(M_i, i) = Q' - X_1 - X_2 - \sum_{i=3}^{5} P(M_i, i), \text{ i.e.}$$

$$Q' = \sum_{i=0}^{5} P(M_i, i) + X_1 + X_2$$

Thus, we have found another preimage for the prehash value Q', namely $M_0 \parallel \ldots \parallel M_5$.

According to the birthday paradox a match is expected if K is chosen to be greater than $\sqrt{|\langle G \rangle|}$. This attack requires $2K$ partial hash computations, which means that it has time complexity $2^{\lceil \frac{m}{2} \rceil + 1}$. For instance, in case of ECOH-224 it allows the computation of second preimages in about 2^{143} steps. Clearly, this is a violation of the SHA-3 competition guidelines that disallow second preimages to be found in less than 2^n time.

It is quite obvious that such an attack has tremendous memory requirements. If, for example, we try to attack the simplest version of ECOH, namely ECOH-224, which makes use of a 283-bit elliptic curve, we must choose K to be at least 2^{142}. Since there are two lists, each of size K, this attack has memory requirements of $2^{143} \cdot 2 \cdot \lceil \frac{283}{32} \rceil \cdot 4 = 2^{146} \cdot \lceil \frac{283}{32} \rceil$ bytes, where $2 \cdot \lceil \frac{283}{32} \rceil \cdot 4$ bytes are necessary for storing a point of a 283-bit curve on a 32 bit platform. Generally speaking, its space complexity is

$$2^{\lceil \frac{m}{2} \rceil + 1} \cdot \left\lceil \frac{m}{b} \right\rceil \cdot \frac{b}{8}$$

bytes on a platform with $b \in \{32, 64\}$ bits. Section 5.5 shows how to get rid of those vast space requirements.

5.5 A Memoryless Implementation of the Ferguson-Halcrow Attack

This section describes a memoryless variant of the Ferguson-Halcrow attack. It gets rid of its tremendous memory requirements by using common cycle finding techniques, which have already been discussed in Section 3.2, at a minimal expense in the running time.

5.5.1 Adapting Floyd's Cycle-Finding Algorithm to Meet in the Middle Attacks

By default, Floyd's cycle-finding algorithm is not applicable to meet-in-the-middle attacks, as the goal is to find collisions between two different functions. However, according to [QD90] there is a way to handle meet-in-the-middle attacks with cycle-finding algorithms. Given two pseudorandom functions f and g, we define another function h:

$$h(M) = \begin{cases} f(M) & \text{if } d(M) = 0, \\ g(M) & \text{otherwise} \end{cases}$$

where $d(\cdot)$ is a decision function outputting 0 or 1 with equal probability. Now, Floyd's algorithm can be applied to $h(\cdot)$ using a random initial value M'. Each round has one of the subsequent outcomes:

1. $(g(M), f(h(M)))$, $(f(M), g(h(M)))$, or
2. $(g(M), g(h(M)))$, $(f(M), f(h(M)))$

Intermediate results of both the first and the second type occur with probability $\frac{1}{2}$. Collisions of the first type are desirable, as our goal is to find collisions between f and g, whereas collisions of the second type are not useful. In such a case the algorithm must be restarted with another random initial value M''.

5.5.2 Expected Run Time

We know already that Floyd's cycle finding algorithm requires $O(\lambda)$ invocations of the hash function and $O(\lambda)$ comparisons in its first loop and $O(\mu)$ hash invocations and $O(\mu)$ comparisons inside the second loop. The additional evaluation of the decision function adds an extra cost of $O(len(x)) = O(1)$ per iteration.

All in all, we get an expected run time of:

$$\underbrace{O(\lambda) \cdot O(H) + O(\lambda) \cdot O(1)}_{\text{loop 1}} + \underbrace{O(\mu) \cdot O(H) + O(\mu) \cdot O(1)}_{\text{loop 2}} =$$

$$= O\left(\sqrt{\frac{2^m \pi}{8}}\right) \cdot O(H) + O\left(\sqrt{\frac{2^m \pi}{8}}\right) + O\left(\sqrt{\frac{2^m \pi}{8}}\right) \cdot O(H) + O\left(\sqrt{\frac{2^m \pi}{8}}\right) =$$

$$= O\left(\sqrt{\frac{2^m \pi}{8}} \cdot H\right) + O\left(\sqrt{\frac{2^m \pi}{8}}\right) = O\left(\sqrt{\frac{2^m \pi}{8}} \cdot H\right),$$

where $O(H)$ denotes the complexity of a partial prehash computation using three input blocks.

5.5.3 Implementation Details

This subsection describes what we have done to implement a memoryless version of the Ferguson-Halcrow attack.

The Attack in Pseudocode

Algorithm 5.3 sketches our implementation in pseudocode. It makes use of the following auxiliary functions:

- $x(M) = M_0 \parallel M_1 \parallel (M_0 \oplus M_1)$, which supplements the exclusive-or block,
- the decision function $d(M) = \bigoplus_{i=0}^{1} (M_i \bmod 2)$,
- the main function $h(\cdot, \cdot)$ based on the idea of Section 5.5.1:

$$h(M, Q) = \begin{cases} \sum_{i=0}^{2} P(M_i, i) & \text{if } d(M) = 0, \\ Q - \left(\sum_{i=0}^{2} P(M_i, i+3) + X_1 + X_2\right) & \text{otherwise,} \end{cases}$$

  where $P(M_i, j)$ maps the message block M_i and the index j to the appropriate curve point, and

Algorithm 5.3 Memoryless Version of the Ferguson-Halcrow Attack

Require: $M = M_0 \parallel M_1$ to be an initial random two-block message and Q' to be the prehash of an arbitrary message M'

1: $M_T = M_H = x(M)$ {The tortoise and the hare have the same initial value}
2: **repeat**
3: $P_T = h(M_T, Q')$ {Compute the tortoise step}
4: $P_H = h(h(M_H, Q'), Q')$ {Compute the hare step}
5: $M_T = x(m(P_T))$ {Compute the message for the next tortoise step}
6: $M_H = x(m(P_H))$ {Compute the message for the next hare step}
7: **until** $P_T = P_H$
8: $M_H = M_T$ {The hare starts from the tortoise's position}
9: $M_T = x(M)$ {The tortoise starts from the beginning}
10: **repeat**
11: $M_{T_1} = M_T$ {Store the current tortoise message}
12: $M_{H_1} = M_H$ {Store the current hare message}
13: $P_T = h(M_T, Q')$ {Compute the tortoise step}
14: $P_H = h(M_H, Q')$ {Compute the hare step}
15: $M_T = x(m(P_T))$ {Compute the message for the next tortoise step}
16: $M_H = x(m(P_H))$ {Compute the message for the next hare step}
17: **until** $P_T = P_H$
18: **if** $d(M_{T_1}) \neq d(M_{H_1})$ **then**
19: **if** $d(M_{T_1}) = 0$ **then**
20: **return** $M = M_{T_1} \parallel M_{H_1}$ {M_{T_1} is the first block of the second preimage M to message M'}
21: **else**
22: **return** $M = M_{H_1} \parallel M_{T_1}$ {M_{H_1} is the first block of the second preimage M to message M'}
23: **end if**
24: **end if**

- the function $m(P) = l_{blen}(x_p) \parallel l_{blen}(y_p)$, which creates a message from a point P that is then used as input for the next invocation of $h(\cdot, \cdot)$, where the function $l_{blen}(x)$ returns the bitstring of the *blen* least significant bits of the binary representation of $x \in \mathbb{F}_{q^m}$.

First Attempt: An Implementation in C

In order to mount the memoryless Ferguson-Halcrow attack against ECOH, we altered ECOH's reference implementation and implemented an 80-bit version of ECOH, called ECOH-80, using a 97-bit binary curve with equation:

$$E : Y^2 + XY = X^3 + X^2 + 1$$

over the binary field $\mathbb{F}_{2^{97}}$. 97 bits were necessary to store away 32 counter bits, 32 index bits and a data block of size 32 bits. This was the smallest curve size we were able to embed into ECOH's reference implementation without compromising it.

Unfortunately, it turned out that ECOH is far too slow for performing such an attack. The evaluation of more than 2^{32} tortoise and hare steps took about three weeks.

Second Attempt: An Implementation in Sage

In order to gain more flexibility in choosing the elliptic curve we decided to implement the attack and the necessary parts of ECOH using Sage, `http://www.sagemath.org`. Sage is Python-based and as such its performance is really bad compared to programs written in the C programming language. Hence, we have chosen $blen = 16$, $ilen = 4$, $clen = 4$ and the subsequent curve

$$Y^2 + XY = X^3 + X^2 + \left(1 + g^6 + g^7\right)$$

over the field $\mathbb{F}_{2^{25}} \simeq \mathbb{F}_2[X]/\left(X^{25} + X^3 + 1\right)$ with g being a generator of $\mathbb{F}^*_{2^{25}}$.

The reader can take a look at our implementation in Listing B.1

Technical Implementation Details As decision function we have chosen:

$$d(M) = \bigoplus_{i=0}^{1} (M_i \bmod 2),$$

where M is an array comprising two message blocks. $d(\cdot)$ decides whether we compute $P_0 + P_1 + P_2$ or $Q - (P_3 + P_4 + P_5 + X_1 + X_2)$. Due to the fact that each round outputs a partial prehash value of size $2 \cdot 25$ bits and the next round's input has to be the size of two blocks (i.e. of size $2 \cdot blen = 32$ bits), we are feeding the lowest 16 bits of both coordinates of the intermediate prehash point as input into the next round.

The Result of Our Attack One can run the attack using the above curve in the following way:

```
1  # Construct a finite field instance
2  F1.<x> = GF(2)[]
3  F.<g> = GF(2^25, name='g', modulus=x^25 + x^3 + 1)
4  # Construct an elliptic curve instance
5  E = EllipticCurve(F, [1, 1, 0, 0, 1+g^6+g^7])
6  # Set ECOH's parameters
7  blen, clen, ilen = 16, 4, 4
8  # Construct an instance of ECOH
9  ecoh = ECOH(E, blen, clen, ilen)
10 # The blocks out of which we create the prehash Q
11 msg_blocks = [0x1D0C, 0x2147, 0x2631]
12 # Create an instance of Attack
13 attack = Attack(ecoh, msg_blocks)
14
15 # Start the attack using the initial message blocks [0x2345, 0x9876]
16 attack.run([0x2345, 0x9876])
```

This computation finishes with the subsequent result:

```
17 MATCH found
18 ([24475, 50920], [8433, 51129])
```

The decision function $d(\cdot)$, which is implemented by `Attack.chooseBranch()`, returns branch 1 for the two message blocks `[24475, 50920]` and branch 0 for the message blocks `[8433, 51129]`. Therefore, we supplement the exclusive-or values yielding blocks 3 and 6 and gain the following second preimage to the message `[0x1D0C, 0x2147, 0x2631]`:

```
[8433, 51129, 59208, 24475, 50920, 39283]
```

which equals

```
[0x20F1, 0xC7B9, 0xE748, 0x5F9B, 0xC6E8, 0x9973]
```

in hexadecimal representation.

5.6 ECOH^2

As already mentioned, the Ferguson-Halcrow attack gives rise to an updated version of ECOH, namely ECOH^2 [Bro09]. ECOH^2 circumvents the previously discussed vulnerability by switching to a curve of the form:

$$Y^2 + XY = X^3 + a_6 \tag{5.1}$$

over the field $\mathbb{F}_{2^d} = \mathbb{F}_{2^{4m}}$. With such quartic extension fields, the original ECOH pseudocode remains largely the same. Using quartic extension fields of the original field seems to be quite unhandy. Instead, Brown et al. show a possibility of handling most operations using two curves over quadratic extension fields.

Algorithm 5.4 sketches ECOH^2 and Table 5.3 lists its updated parameters. Note that compared to ECOH, lines 8 and 9 of the algorithm have changed and that parameters E and G have been replaced by the finite field's extension degree d.

5.6.1 The Representation of Elements of the Extension Fields

In order to represent elements of the extension fields by means of elements of $\mathbb{F}_{2^m}$, we can utilize so-called tower representations.

The Tower Representation in $\mathbb{F}_{2^{2m}}$

For $\mathbb{F}_{2^{2m}}$ being an extension field of $\mathbb{F}_{2^m}$, we are able to represent elements of it in the following manner:

$$x = x_0 + x_1 u,$$

where $x_0, x_1 \in \mathbb{F}_{2^{2m}}$ are given in their polynomial representation

$$x_i = c_{m-1}^{(i)}\alpha^{m-1} + \ldots + c_1^{(i)}\alpha^1 + c_0^{(i)} \;\; i \in \{0,1\}$$

and $u \in \mathbb{F}_4$ such that $u^2 + u + 1 = 0$. This representation is called (a, u) *tower representation*. The bitstring representation of x is $x_0 \parallel x_1$, where the rightmost bit of x_0 is said to be the *constant bit*.

The Tower Representation in $\mathbb{F}_{2^{4m}}$

Likewise, for $\mathbb{F}_{2^{4m}}$ being an extension field of $\mathbb{F}_{2^m}$, we are able to represent elements of it in the following manner:

$$x = (x_{10} + x_{11}u)v + (x_{00} + x_{01}u),$$

where $x_{00}, x_{01}, x_{10}, x_{11} \in \mathbb{F}_{2^m}$ are given in their polynomial representation

$$x_{ij} = c_{m-1}^{(ij)}\alpha^{m-1} + \ldots + c_1^{(ij)}\alpha^1 + c_0^{(ij)} \;\; i,j \in \{0,1\}$$

and both $u \in \mathbb{F}_4$ and $v \in \mathbb{F}_{16}$ satisfying the subsequent equations:

$$u^2 + u + 1 = 0$$
$$v^2 + v + u = 0$$

This representation is called (α, u, v) *tower representation*. The bitstring representation of x is $x_{10} \parallel x_{11} \parallel x_{00} \parallel x_{01}$, where the rightmost bit of x_{00} is said to be the *constant bit*.

5.6.2 Curve Definitions in Detail

It is beneficial with respect to the throughput of ECOH^2 to implement ECOH^2 using a curve and its quadratic twist over $\mathbb{F}_{2^{2m}}$. Both of those curves can be seen as subgroups of Curve 5.1 over the quartic extension field. This subsection lists these two equivalent approaches.

The Curve over the Quartic Extension Field

The Curve 5.1 is chosen in such a way that it has $8rs$ $\mathbb{F}_{2^{4m}}$-rational points for two primes r and s. The curve's base point G is chosen to be $\mathbb{F}_{2^{4m}}$-rational, of order rs and for reasons of performance lexicographically least with respect to its bitstring representation.

The Curve and its Twist over the Quadratic Extension Field

In order to tweak ECOH^2's performance we can also operate on the curve

$$Y^2 + XY = X^3 + a_6 \tag{5.2}$$

over the quadratic extension field $\mathbb{F}_{2^{2m}}$ and its quadratic twist over $\mathbb{F}_{2^{2m}}$ given by

$$Y^2 + XY = X^3 + uX^2 + a_6 \tag{5.3}$$

with $u \in \mathbb{F}_4$, $a_6 = 1 + uw \in \mathbb{F}_{2^{2m}}, w \in \mathbb{F}_{2^{2m}}$ and the bitstring representation of a_6 chosen lexicographically least among the curves

- with Equation (5.2) having $4r$ rational points in $\mathbb{F}_{2^{2m}}$, where $r \in \mathbb{P}$, and
- with twisted Equation (5.3) having $2s$ rational points in $\mathbb{F}_{2^{2m}}$, where $s \in \mathbb{P}$.

5.6.3 Using the Twist

As [Bro09, Section 4.3] points out, for roughly one half of all x-coordinates in $\mathbb{F}_{2^{2m}}$ there is a valid y-coordinate in $\mathbb{F}_{2^{2m}}$. Consequently, in such a case we can avoid calculations in the quartic extension field $\mathbb{F}_{2^{4m}}$. In all other cases the y-coordinate of some point $P = (x_0, y_0)$ is not an element of $\mathbb{F}_{2^{2m}}$. But, it is possible to remap P another point lying on the twist and continue with our operations on this curve.

5.6.4 A Word on ECOH^2's Efficiency

The usage of larger finite fields suggests that ECOH^2's performance drops even more. However, as [Bro09, Section 4] points out, most operations can be achieved over the subfield $\mathbb{F}_{2^{2m}}$ of $\mathbb{F}_{2^{4m}}$', except for the final two steps of Algorithm 5.4. One advantage over ECOH seems to be its larger block length, which means that more amount of data can be processed in each step. They come to the conclusion that ECOH^2 should perform slightly better than ECOH on long messages.

Algorithm	n	d	$blen$	$ilen$	$clen$
ECOH^2-224	224	1132	384	64	64
ECOH^2-256	256	1132	384	64	64
ECOH^2-384	384	1636	640	64	64
ECOH^2-512	512	2284	768	128	128

Table 5.3: The four different versions of ECOH^2 [Bro09, Table 1]

Algorithm 5.4 Generic ECOH^2 pseudocode [Bro09, Table 2]

Require: Message M of maximum bitlength $len(M) \leq 2^{ilen} - 1$.

1: Set $N = M \parallel 1 \parallel 0^j$ with j chosen minimally, such that $blen | len(N)$.
2: Split N into k blocks of bitlength $blen$: $N_0, \ldots, N_{k-1}$.
3: **for** $i = 0$ to $i = k - 1$ **do**
4: Index block N_i: $O_i = N_i \parallel I_i$, where I_i is the bit-representation of integer i of length $ilen$ bits.
5: **end for**
6: Compute the checksum block $O_k = \left(\bigoplus_{i=0}^{k-1} N_i\right) \parallel I_{len(M)}$, where $I_{len(M)}$ stands for the $ilen$-bit-representation of the message bitlength $len(M)$.
7: **for** $i = 0$ to $i = k$ **do**
8: Find bit string $X_i = \left(0^{d-(blen+ilen+clen)} \parallel O_i \parallel 0^{clen}\right) \oplus C_i$, where C_i is of length d and chosen minimally such that X_i belongs to a valid x-coordinate x_i of an element of the elliptic curve group $\langle G \rangle$.
9: Decompress point $P = (x_i, y_i)$ such that the leftmost bit of N_i equals the constant bit of $y_i \cdot x_i^{-1}$.
10: **end for**
11: Let $Q = \sum_{i=0}^{k} P_i$.
12: **return** n-bit representation of $\lfloor x_Q + \lfloor x_Q/2 \rfloor G)/2 \rfloor \bmod 2^n$.

Part III

Arithmetic Speedups

Chapter 6

Edwards Curves

One big disadvantage of conventional elliptic curves is the inconvenient addition law. On the one hand its performance is quite bad and on the other hand it is unhandy as it consists of two separate methods: the chord method and the tangent method. Recall that the addition of two distinct points P, Q is performed via the chord method, whereas a point P is doubled via the tangent method. In this way one always has to make a distinction between doubling and adding, which does not only lower the overall performance but also opens the way for side-channel attacks and requires more precious space on hardware implementations.

Fortunately, in 2007 Harold M. Edwards had come up with a new approach, which was later named Edwards curves and Edwards coordinates, respectively [Edw07]. Edwards curves are birationally equivalent to ordinary Weierstrass curves. Moreover, the addition law valid on Edwards curves is able to resolve both points of criticism mentioned above. It is astonishingly fast and complete as well. The latter signifies that there is a single summation formula applicable to all possible pairs of points including the point at infinity. In particular this means that no distinction is drawn between doubling and addition. Thereby, all the case differentiations in the ordinary Weierstrass addition (see Algorithm 2.1) render obsolete. One drawback is, however, that only a fraction of all Weierstrass curves can be captured by Edwards curves over the same field.

Since 2007 Daniel J. Bernstein and Tanja Lange have extended the theoretical background on Edwards coordinates [BL07a, BL07b, BL07c, BLF08, BBJ$^+$08]. They found various tweaks to improve the addition performance even more and were also able to identify an extended curve type called twisted Edwards curves [BBJ$^+$08], which expands the number of curves corresponding to Weierstrass curves. Furthermore, in 2008, Bernstein introduced so-called binary Edwards curves [BLF08].

This chapter starts with an introduction to Edwards coordinates and then expands on the theoretical extensions of this theory. We are also going to discuss Edwards curves over binary fields as well as the generalized notion of twisted Edwards curves.

6.1 Original Edwards Coordinates

What Edwards showed in [Edw07] was that all elliptic curves over algebraic field extensions of the rational numbers $\mathbb{Q}$, so-called number fields, can be converted to the form:

$$\boxed{X^2 + Y^2 = c^2(1 + X^2Y^2)} \tag{6.1}$$

with neutral element $(0, c)$ and a simple, symmetric addition law:

$$P + Q = (x_1, y_1) + (x_2, y_2) =$$
$$\Big((x_1y_2 + y_1x_2) \cdot (c(1 + x_1x_2y_1y_2))^{-1}, (y_1y_2 - x_1x_2) \cdot (c(1 - x_1x_2y_1y_2))^{-1}\Big)$$

The inverse of a point $P = (x_1, y_1)$ on the curve is defined as $-P = (-x_1, y_1)$. What is important is that this addition instruction is *complete*. In other words, it can handle any kinds of input points.

6.2 Edwards Coordinates according to Bernstein and Lange

In 2007, Bernstein and Lange were able to extend Edwards' curve shape to finite fields $\mathbb{F}_q$ with $char(\mathbb{F}_q) \neq \{2, 3\}$ [BL07b] and hence to make this notion relevant for cryptography. This section explains this approach by means of a slightly extended curve form. Later we are going to see, how curves over finite fields of characteristic 2 can also be brought to a form having similar properties. Note, however, that although the following descriptions can also be applied to curves over fields of characteristic 3, we omit this case since this thesis does at no point deal with such curves. Also note that we are going to discuss the following statements with respect to finite fields. Except for Theorem 6.1, all of the following propositions remain valid for general fields with characteristic not 2. See [BL07b, Theorem 2.1] for a full and generic version of Theorem 6.1.

In [BL07b], Bernstein and Lange suggest a minor modification of Equation (6.1) in order to cover more Weierstrass curves over an arbitrary finite field $\mathbb{F}_q$ with $char(\mathbb{F}_q) \neq \{2, 3\}$:

$$\boxed{E^{E,d} : X^2 + Y^2 = 1 + dX^2Y^2 \text{ with } d \in \mathbb{F}_q \setminus \{0, 1\}} \tag{6.2}$$

If $d = d'c'^4$ for $d' \in \mathbb{F}_q^*$, this curve is isomorphic to:

$$\boxed{E^{E,c',d'} : X'^2 + Y'^2 = c'^2(1 + d'X'^2Y'^2) \text{ with } X' = c'X \text{ and } Y' = c'Y} \tag{6.3}$$

which resembles the original equation. Evidently, Edwards curves are symmetric. Hence, if $P = (x_0, y_0)$ is a valid point, $P' = (y_0, x_0)$ also is.

Example 6.1. The curve $X^2 + Y^2 = 1 + 2X^2Y^2$ defined over $\mathbb{F}_{11}$ consists of the points $(0, 1), (0, 10), (1, 0), (3, 4), (3, 7), (4, 3), (4, 8), (7, 3), (7, 8), (8, 4), (8, 7)$, and $(10, 0)$.

6.2.1 Birational Equivalence

Curve 6.2 is birationally equivalent over $\mathbb{F}_q$ to a quadratic twist of some Weierstrass curve E. If, additionally, there is a unique point of order 2 on E, then the constant $d \in \mathbb{F}_q$ in Equation (6.2) is a non-square. These facts are summarized in the theorem below.

Theorem 6.1 (Characterization of the birational equivalence)**.** *Let $\mathbb{F}_q$ be a finite field having $char(\mathbb{F}_q) \notin \{2, 3\}$, $E/\mathbb{F}_q$ be an elliptic curve such that $E(\mathbb{F}_q)$ comprises an element of fourth order. If $E(\mathbb{F}_q)$ comprises exactly one element P with $ord(P) = 2$ then there is a non-square $d \in \mathbb{F}_q$ such that E is birationally equivalent to the curve $X^2+Y^2 = 1+dX^2Y^2$ over $\mathbb{F}_q$.*

Proof. We refer to [BL07b, Theorem 2.1]. □

An explicit mapping rule between Montgomery-type elliptic curves and Edwards curves is given in Theorem 6.3.

6.2.2 The Addition Law

This subsection presents and discusses the addition law belonging to Curve 6.3. Like above, we are given a finite field $\mathbb{F}_q$ with $char(\mathbb{F}_q) \notin \{2,3\}$, two coefficients $c, d \in \mathbb{F}_q^*$ with $dc^4 \neq 1$. Then, the addition law on Curve 6.3 looks as follows:

$$P + Q = (x_1, y_1) + (x_2, y_2) =$$
$$\Big((x_1y_2 + y_1x_2) \cdot (c(1 + dx_1x_2y_1y_2))^{-1}, (y_1y_2 - x_1x_2) \cdot (c(1 - dx_1x_2y_1y_2))^{-1}\Big)$$

The inverse of some point $P = (x_0, y_0)$ is $-P = (-x_0, y_0)$ and the neutral element is $(0, c)$. The point $(0, -c)$ has order 2 and $(c, 0)$ as well as $(-c, 0)$ have order 4. Note that these results can be shown by simple calculations.

Example 6.2. We are given the curve from Example 6.1. Then, the elements $(1, 0)$ and $(10, 0)$ have order 4 and are inverse to each other. Obviously, the neutral element is $(0, 1)$ and the point $(0, 10)$ has order 2. Next, we are going to add the points $(3, 4)$ and $(4, 8)$:

$$x_3 = 40 \cdot 769^{-1} \equiv 7 \cdot 10^{-1} \bmod 11 \equiv 4 \bmod 11$$
$$y_3 = 20 \cdot (-767)^{-1} \equiv 9 \cdot 4 \bmod 11 \equiv 3 \bmod 11$$

and obtain the result $(4, 3)$.

The theorem below ensures that the sum of two points is on the curve as long as the denominator in both components does not vanish. That is, the term $dx_1x_2y_1y_2 \neq \pm 1$.

Theorem 6.2 (Correctness, [BL07b, Theorem 3.1])**.** *Let $\mathbb{F}_q$ be a finite field with $char(\mathbb{F}_q) \notin \{2, 3\}$, $c, d \in \mathbb{F}_q^*$ with $dc^4 \neq 1$ and $P = (x_1, y_1), Q = (x_2, y_2)$ be points on Curve 6.3. Under the assumption that $dx_1x_2y_1y_2 \neq \pm 1$, $P + Q = R = (x_3, y_3)$ lies on the curve.*

Proof. What we want to show is that $R = (x_3, y_3)$, which is determined through the addition law, fulfills $x_3^2 + y_3^2 = c^2(1 + dx_3^2y_3^2)$. To do so, we consider the term $x_3^2 + y_3^2 - c^2dx_3^2y_3^2$ and plug the addition law in:

$$x_3^2 + y_3^2 - c^2dx_3^2y_3^2 =$$
$$= (x_1y_2 + y_1x_2)^2 (c(1 + dx_1x_2y_1y_2))^{-2} + (y_1y_2 - x_1x_2)^2 (c(1 - dx_1x_2y_1y_2))^{-2} -$$
$$c^2d (x_1y_2 + y_1x_2)^2 (y_1y_2 - x_1x_2)^2 (c(1 + dx_1x_2y_1y_2))^{-2} (c(1 - dx_1x_2y_1y_2))^{-2} =$$

which simplifies to:

$$= (\underbrace{x_1^2 + y_1^2}_{c^2(1+dx_1^2y_1^2)} - \underbrace{(x_2^2 + y_2^2)dx_1^2y_1^2}_{c^2(1+dx_2^2y_2^2)dx_1^2y_1^2})(\underbrace{x_2^2 + y_2^2}_{c^2(1+dx_2^2y_2^2)} - \underbrace{(x_1^2 + y_1^2)dx_2^2y_2^2}_{c^2(1+dx_1^2y_1^2)dx_2^2y_2^2}) \left(c(1 - (dx_1x_2y_1y_2)^2)\right)^{-2} =$$
$$= \left(c^2(1 - (dx_1x_2y_1y_2)^2)\right)^2 \left(c(1 - (dx_1x_2y_1y_2)^2)\right)^{-2} = c^2$$

Hence, we have shown: $x_3^2 + y_3^2 = c^2 \left(1 + dx_3^2y_3^2\right)$. □

The next theorem makes sure that the above addition law corresponds to the addition law on a birationally equivalent Weierstrass curve E.

Theorem 6.3 (Compliance with Weierstrass addition)**.** *Under the same assumptions as in Theorem 6.2, we define the following elliptic curve (in Montgomery form):*

$$E^{M_{e,f}} : eY^2 = X^3 + fX^2 + X$$

with parameters $e = (1 - dc^4)^{-1}$ *and* $f = 4e - 2$*. Besides, we are given the following birational equivalence between the Edwards curve* $E^{E_{c,d}}$ *and* $E^{M_{e,f}}$*:*

$$\begin{array}{rccc} \phi: & E^{E_{c,d}} & \to & E^{M_{e,f}} \\ & (x_0, y_0) & \mapsto & (\alpha(y_0), 2c \cdot \alpha(y_0) \cdot x_0) \end{array}$$

where $\alpha(x) = (c + x)(c - x)^{-1}$*,* $\phi(0, c) = \mathcal{O}$ *and* $\phi(0, -c) = (0, 0)$*.*

Then, for $P, Q, R \in E^{E_{c,d}}(\mathbb{F}_q)$ *with* $P + Q = R$ *we have* $\phi(P), \phi(Q) \in E^{M_{e,f}}(\mathbb{F}_q)$ *as well as* $\phi(P) + \phi(Q) = \phi(R) \in E^{M_{e,f}}(\mathbb{F}_q)$*.*

Proof. See [BL07b, Theorem 3.2]. □

From this it follows that we can operate on an Edwards curve with extra costs of one initial evaluation and one final inversion of the above point correspondence for each batch operation.

Example 6.3. We are given the curve from Example 6.1. Then, $e = (1-2)^{-1} = (-1)^{-1} \equiv 10 \bmod 11$ and $f = 38 \equiv 5 \bmod 11$. Hence, it is birationally equivalent to the Montgomery curve $10Y^2 = X^3 + 5X^2 + X$.

By now, we do not know under what conditions the addition law is valid. The next theorem states that the addition law is defined as long as d is not a square. In other words it is *complete*.

Theorem 6.4 (Completeness, [BL07b, Theorem 3.3])**.** *Under the same conditions as in Theorem 6.3, let* d *be a non-square and let* $P = (x_1, y_1), Q = (x_2, y_2)$ *be points of the Edwards Curve 6.3 with parameters* c *and* d*. Then, the denominator of the Edwards addition formula is non-zero, meaning that* $dx_1x_2y_1y_2 \neq \pm 1$*.*

Proof. We prove this statement by contradiction. Suppose that d is a non-square and that $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are two exceptional points, i.e. one denominator in the addition law vanishes for P and Q, which can only be the case when $\epsilon = dx_1x_2y_1y_2 \in \{\pm 1\}$. This implies that $x_1, x_2, y_1, y_2 \neq 0$. As the point Q fulfills Equation (6.3), we have $x_2^2 + y_2^2 = c^2(1 + dx_2^2y_2^2)$ and get

$$dx_1^2y_1^2(x_2^2 + y_2^2) = c^2(dx_1^2y_1^2 + d^2x_2^2y_2^2x_1^2y_1^2) = c^2(dx_1^2y_1^2 + \epsilon^2) = c^2(dx_1^2y_1^2 + 1) = x_1^2 + y_1^2$$

through multiplication with $dx_1^2y_1^2$. With this piece of information the term $(x_1 + \epsilon y_1)^2 = x_1^2 + y_1^2 + 2\epsilon x_1y_1$ can be transformed to $dx_1^2y_1^2(x_2 + y_2)^2$. Now, if $x_2 \pm y_2 \neq 0$, we get $d = ((x_1 \pm \epsilon y_1)/(x_1y_1(x_2 \pm y_2)))^2$. Thus, d is a square, which is a contradiction. Otherwise, if $(x_2 + y_2, x_2 - y_2) = (0, 0)$ we have $(x_2, y_2) = (0, 0)$, contradiction. □

From now on, we refer to Edwards curves with complete addition law as *complete Edwards curves*.

Remark 6.1. Note that although Edwards curves feature a complete addition law, explicit doubling formulas still exist for the simple reason that some field multiplications can be replaced by squarings, which have smaller costs in general.

Example 6.4. The curve given in Example 6.1 is complete since 2 is a non-square in $\mathbb{F}_{11}$.

Note that for finite fields this curve type covers at least one fourth of all non-binary Weierstrass curves. This amount of birationally equivalent ordinary curves over finite fields is bounded because of the necessary existence of a point of order 4.

6.3 Twisted Edwards Curves

As we have heard already, there is one major drawback of ordinary complete Edwards curves. Namely, the number of such curves is limited by the requirement for a Weierstrass curve to have an element of order four in order to be convertible to Edwards form. Hence, it is estimated that only about one quarter of all Weierstrass curves over finite fields $\mathbb{F}_q$ with $char(\mathbb{F}_q) \notin \{2, 3\}$ have corresponding complete Edwards curves. Daniel J. Bernstein et al. in [BBJ+08], introduced a generalization of Edwards curves, namely twisted Edwards curves, to deal with that issue by using twists.

This section starts with a definition of twisted Edwards curves and then shows their most important properties such as their correspondence with Montgomery curves. Although, this section focusses on finite fields, all of the statements also apply for general fields of characteristic not 2.

Definition 6.1 (Twisted Edwards curve)**.** Let $\mathbb{F}_q$ be a finite field with $char(\mathbb{F}_q) \notin \{2, 3\}$ and let $a, d \in \mathbb{F}_q^*$ with $a \neq d$. Then, the *twisted Edwards curve with coefficients* a, d is defined as:

$$\boxed{E^{E_T,a,d} : aX^2 + Y^2 = 1 + dX^2Y^2} \tag{6.4}$$

with j-invariant $j(E^{E_T,a,d}) = 16(a^2 + 14ad + d^2)^3(ad(a-d)^4)^{-1}$.

For $a = 1$ this coincides with the notion of ordinary non-binary Edwards curves. Moreover, the curve $E^{E,a,d}$ is a quadratic twist of the curve

$$E^{E,1,da^{-1}} : X'^2 + Y'^2 = 1 + (da^{-1})X'^2Y'^2$$

over the field $\mathbb{F}_q(\sqrt{a})$ defined through the isomorphism

$$(X', Y') \mapsto (a^{-\frac{1}{2}}X', Y').$$

Note that $\mathbb{F}_q(\sqrt{a})$ and $\mathbb{F}_q$ are the same in case that $\sqrt{a} \in \mathbb{F}_q$.

6.3.1 Montgomery Curves and Twisted Edwards Curves

This section explains the reason why the notion of Montgomery curves and the notion of twisted Edwards curves are the same. Recall that a Montgomery curve over $\mathbb{F}_q$ with coefficients $e, f \in \mathbb{F}_q$ with $e \neq \pm 2, f \neq 0$ is defined by the equation:

$$E^{M,e,f} : fY^2 = X^3 + eX^2 + X.$$

The next theorem states that both curve sets are the same.

Theorem 6.5. *Let $\mathbb{F}_q$ be a finite field with $char(\mathbb{F}_q) \notin \{2, 3\}$. Then the subsequent statements hold:*

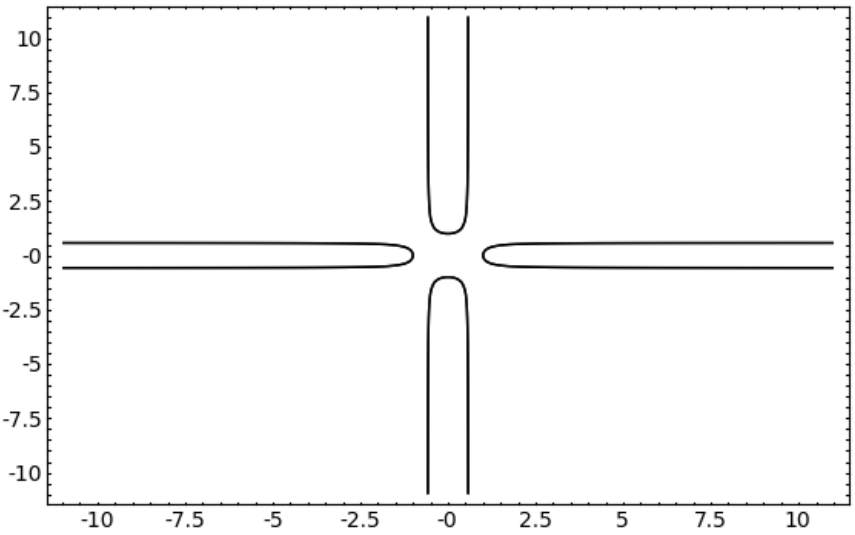

Figure 6.1: Plot of the Edwards curve $X^2 + Y^2 = 1 + 3X^2Y^2$ over $\mathbb{R}$

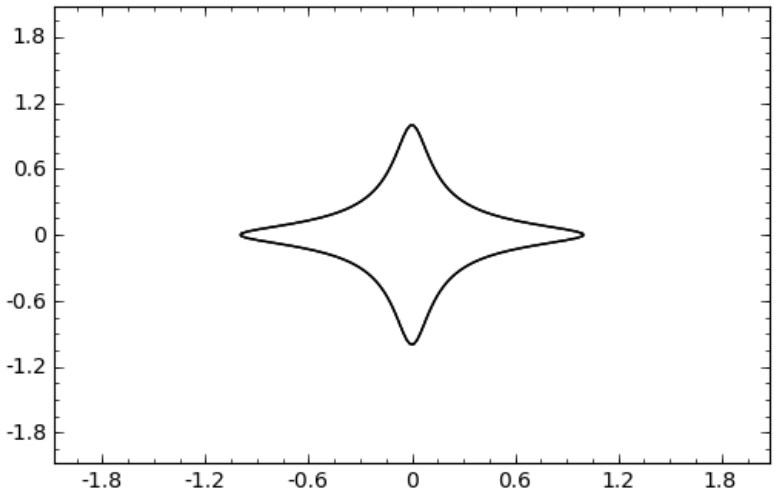

Figure 6.2: Plot of the Edwards curve $X^2 + Y^2 = 1 - 100X^2Y^2$ over $\mathbb{R}$

- *Each twisted Edwards curve over $\mathbb{F}_q$ with coefficients $a, d \in \mathbb{F}_q, a \neq d$ is birationally equivalent over $\mathbb{F}_q$ to some Montgomery curve. The birational equivalence is defined by the map:*

$$\begin{array}{rccc} \phi: & E^{E_T,a,d} & \to & E^{M,e,f} \\ & (x_0, y_0) & \mapsto & (\alpha(y_0), x_0 \cdot \alpha(y_0)) \end{array}$$

  with $\alpha(y) = (1+y)(1-y)^{-1}$ and coefficients a, d mapped to $(e, f) = (2(a+d)(a-d)^{-1}, 4(a-d)^{-1})$.

- *On the flipside, each Montgomery curve over $\mathbb{F}_q$ with coefficients $e \in \mathbb{F}_q \setminus \{\pm 2\}$ and $f \in \mathbb{F}_q^*$ is birationally equivalent over $\mathbb{F}_q$ to some twisted Edwards curve. The birational equivalence is defined by the inverse map ϕ^{-1}:*

$$\begin{array}{rccc} \phi^{-1}: & E^{M,e,f} & \to & E^{E_T,a,d} \\ & (x_0', y_0') & \mapsto & (x_0' y_0'^{-1}, (x_0'-1)(x_0'+1)^{-1}) \end{array}$$

  and the coefficients e, f are mapped to $(a, d) = ((e+2)f^{-1}, (e-2)f^{-1})$.

Proof. See [BBJ+08, Theorem 3.2]. □

ϕ has two exceptional points (cf. Section 2.4.4), namely the point $(0, -1)$ of order 2 and the neutral element $(0, 1)$. Hence, the first corresponds to the second order point $(0, 0)$, which is part of any arbitrary Montgomery curve, whereas the latter corresponds to $\mathcal{O} \in E^{M,e,f}$. For a comprehensive listing of all exceptional points of ϕ^{-1} take a look at [BBJ+08, Section 3].

6.3.2 More Curves through Isogenies

If a Weierstrass curve is neither isomorphic to an Edwards curve nor to a twisted Edwards curve, there is still the possibility that it is isogenous to a twisted Edwards curve. In [BBJ+08] Bernstein and others show that each Weierstrass curve having three points of order two is 2-isogenous to some curve in twisted Edwards form:

Theorem 6.6. *Let $\mathbb{F}_q$ be a field with $char(\mathbb{F}_q) \notin \{2, 3\}$. Each Weierstrass curve $E/\mathbb{F}_q$ with three $\mathbb{F}_q$-rational points of second order, namely $(0,0), (u_1, 0), (u_2, 0)$, is 2-isogenous over $\mathbb{F}_q$ to the curve $E^{E_T,4u_1,4u_2}$ in twisted Edwards shape.*

Proof. See [BBJ+08, Theorem 5.1]. □

6.3.3 The Addition Law

The addition law for twisted Edwards curves with parameters a and d over a finite field $\mathbb{F}_q$ having $char(\mathbb{F}_q) \notin \{2, 3\}$ looks as follows:

$$P + Q = (x_1, y_1) + (x_2, y_2) =$$
$$\left((x_1 y_2 + y_1 x_2) \cdot (1 + d x_1 x_2 y_1 y_2)^{-1}, (y_1 y_2 - a x_1 x_2) \cdot (1 - d x_1 x_2 y_1 y_2)^{-1}\right)$$

This formula strongly resembles the formula valid on ordinary curves with short Equation (6.2). Just like in this case, the neutral element is $(0, 1)$ and the inverse of $P = (x_0, y_0)$ is the point $(-x_0, y_0)$. Suppose $a \in \mathbb{F}_q$ is a square. Then it follows that the twisted Edwards curve $E^{E_T,a,d}$ over $\mathbb{F}_q$ is isomorphic over $\mathbb{F}_q$ to an Edwards curve $E^{E,1,da^{-1}}$. Thus, the results presented in Section 6.2 remain valid. This implies that the addition law for twisted Edwards curves is complete if da^{-1} is a non-square, i.e. if d is a non-square.

6.4 Binary Edwards Curves

We are now interested in Edwards curves over binary fields. In [BLF08], Bernstein and Lange describe a curve equation that allows us to expand the notion of Edwards curves to base fields of characteristic 2. Unfortunately, unlike the addition law on non-binary Edwards curves the group law on binary Edwards curves does not bring along new speed records. Although, we have to point out that performance improvements, especially with regard to doubling, coming very close to current speed records do exist. For further details on that, we refer to the Explicit-Formulas Database, `http://hyperelliptic.org/EFD`.

In this section we deal with this new type of Edwards curves. Besides to discussing the actual curve features, we take a look at its addition law, the birational equivalence and the like. Note that we are going to discuss the following statements with respect to finite fields of the type $\mathbb{F}_{2^n}$. Except for Theorem 6.8, all of the subsequent propositions remain valid for general fields with characteristic 2.

Definition 6.2 (Binary Edwards curve)**.** We assume $\mathbb{F}_{2^n}$ to be a field of characteristic 2 and $d_1 \neq 0, d_2$ to be elements of $\mathbb{F}_{2^n}$ with $d_2 \neq d_1^2 + d_1$. Then, the *binary Edwards curve with coefficients d_1 and d_2* is given by:

$$\boxed{E^{E_B,d_1,d_2} : d_1(X+Y) + d_2(X^2+Y^2) = (X+X^2)(Y+Y^2)} \tag{6.5}$$

In comparison with Equation (6.3) the new equation appears quite awkward. Nonetheless, both curve types have the same main properties in common. From Equation (6.3) one can read that the curve is symmetric with respect to X and Y. Hence, if $P = (x_0, y_0)$ is a valid point of E^{E_B}, $P' = (y_0, x_0)$ also is.

Also noteworthy is the subsequent statement:

Theorem 6.7 (Nonsingularity)**.** *Every binary Edwards curve is non-singular.*

Proof. See [BLF08, Theorem 2.2]. □

6.4.1 Birational Equivalence

Curve 6.5 is birationally equivalent to the subsequent non-supersingular binary elliptic curve:

$$E : Y'^2 + X'Y' = X'^3 + a_2X'^2 + a_6$$

with coefficients $a_2 = d_1^2 + d_2$ and $a_6 = d_1^4(d_1^4 + d_1^2 + d_2^2)$ and j-invariant $j(E) = a_6^{-1}$. The birational equivalence is given by the map

$$\begin{array}{rccc} \phi: & E^{E_B,d_1,d_2} & \to & E \\ & (x_0, y_0) & \mapsto & (x_0', y_0') \end{array}$$

with $\phi(0,0) = \mathcal{O}$ and

$$x_0' = d_1\alpha(x_0 + y_0)(x_0y_0 + d_1(x_0 + y_0))^{-1},$$
$$y_0' = d_1\alpha(x_0(x_0y_0 + d_1(x_0 + y_0))^{-1} + d_1 + 1)$$

otherwise; where $\alpha = d_1^2 + d_1 + d_2$. The inverse map $\phi^{-1} : (x_0', y_0') \mapsto (x_0, y_0)$ is defined followingly:

$$x_0 = d_1(x_0' + \alpha)(x_0' + y_0' + \alpha\beta)^{-1},$$
$$y_0 = d_1(x_0' + \alpha)(y_0' + \alpha\beta)^{-1}$$

with $\phi^{-1}(\mathcal{O}) = (0,0)$ and $\beta = d_1^2 + d_1$.

6.4.2 The Addition Law

This subsection is devoted to the addition law for binary Edwards curves and its main properties.

The sum of two elements $P = (x_1, y_1), Q = (x_2, y_2) \in E^{E_B,d_1,d_2}$ is defined as:

$$P + Q = (x_1, y_1) + (x_2, y_2) =$$
$$\Big((\alpha + \beta(x_1, x_2, y_1, y_2))(d_1 + \gamma(x_1)\delta)^{-1}, (\alpha + \beta(y_1, y_2, x_1, x_2))(d_1 + \gamma(y_1)\delta)^{-1}\Big)$$

where

- $\alpha = d_2(x_1 + y_1)\delta$,
- $\beta(a, b, c, d) = d_1(a + b) + \gamma(a)\,(b\,(c + d + 1) + cd)$,
- $\gamma(a) = a + a^2$, and
- $\delta = x_2 + y_2$.

This yields a valid point of the curve, when the denominators of both components do not vanish.

From the addition formulae it follows that the neutral element is the point $(0, 0)$ and that (y_0, x_0) is inverse to (x_0, y_0). Another interesting property of the addition instruction is that $(x_0, y_0) + (1, 1)$ always yields $(x_0 + 1, y_0 + 1)$. This implies that the point $(1, 1)$ is of order 2, meaning that $(1, 1) + (1, 1) = (0, 0)$.

Theorem 6.8 (Compliance with Weierstrass addition)**.** *Let $\mathbb{F}_{2^n}$ be a finite field, $d_1 \neq 0, d_2 \in \mathbb{F}_{2^n}$ with $d_2 \neq d_1 + d_1^2$ and E^{E_B,d_1,d_2} be a binary Edwards curve with coefficients d_1, d_2. Let $P, Q \in E^{E_B,d_1,d_2}(\mathbb{F}_{2^n})$ and $P + Q = R \in E^{E_B,d_1,d_2}(\mathbb{F}_{2^n})$. Then, we have $\phi(P) + \phi(Q) = \phi(R)$.*

Proof. See [BLF08, Theorem 3.5]. □

The remainder of this subsection deals with the notion of completeness. We are going to see that if d_2 is not of the form $t^2 + t$, the addition law is complete and that for almost all n binary Weierstrass curves have complete binary Edwards counterparts.

Theorem 6.9 (Completeness)**.** *Fix a field $\mathbb{F}_{2^n}$, $d_1 \neq 0, d_2 \in \mathbb{F}_{2^n}$ and E^{E_B,d_1,d_2} a binary Edwards curve with coefficients d_1 and d_2. In addition, we assume that there is no element $t \in \mathbb{F}_{2^n}$ such that $d_2 = t^2 + t$. Given that, the addition law on $E^{E_B,d_1,d_2}(\mathbb{F}_{2^n})$ is complete.*

Proof. See [BLF08, Theorem 4.1]. □

From now on, we call such curves *complete binary Edwards curves.*

Theorem 6.10. *Fix $n \in \mathbb{N}_{>2}$. Then it follows that for every Weierstrass curve $E/\mathbb{F}_{2^n}$ exists a birationally equivalent complete binary Edwards curve over $\mathbb{F}_{2^n}$.*

Proof. See [BLF08, Theorem 4.3]. □

This means that unlike complete Edwards curves over fields of prime characteristic, complete binary Edwards curves cover all Weierstrass curves over binary fields $\mathbb{F}_{2^n}$ when $n > 2$.

NIST curve	d_1	d_2
B-163	0x04	0x6ac25b85badf8927593d21c366da89c03969f3494
B-233	0x02	0x61fe1589ee5e1d39d1fb8c781b5c72abba94bc8494f97e51b41876a448
B-283	0x08	0x6dcaf32715e4ac7afa0660c227edbe0d68bf48828c11093285f020ab85d10abb7a6eb70

Table 6.1: Edwards coefficients of some binary NIST curves in hexadecimal representation

6.5 Converting Standardized Curves to Twisted Edwards Form

Recall that in order to transform a Weierstrass curve E to Edwards shape, E must have a point of order four. For E being convertible to twisted Edwards form, still requires E to have a point of second order, why the cofactor h of E must be divisible by two. For safety's sake, however, cofactors greater than four are deprecated. And so, our only chance is to find standardized curves with $h = 2$ or $h = 4$.

There are several standards, like [Sta00], [Nat00] or [Int05], suggesting curves with strong cryptographic properties. But, only the *Standards for Efficient Cryptography 2 (SEC2)* [Sta00] includes two curves, namely *sec112r2* with $p \approx 2^{112}$ and *sec128r2* with $p \approx 2^{128}$, having cofactor $h = 4$. Unfortunately, also the SEC2 standard does not suggest appropriate curves over larger prime fields and these curve sizes can already be considered to be outdated. So, the only remaining way of converting curves with $h = 1$ to Edwards form and twisted Edward form, respectively, is to replace the prime field $\mathbb{F}_p$ by an extension field $\mathbb{F}_{p^n}$ so that $E(\mathbb{F}_{p^n})$ comprises an element of order 4 and 2, respectively. Yet, we can not recommend this approach as the increased field size renders the performance benefits of Edwards curves void.

In contrast to prime field curves, any standardized binary curve can be transformed to the binary Edwards shape. Note, however, that binary Edwards curves are not the fastest binary curves available. Still, the possibility to beat off side-channel attacks and the reduced space of hardware implementations may make them attractive.

Example 6.5. Table 6.1 shows the Edwards coefficients of the binary NIST-curves B-163, B-233, B-283 (cf. [Nat00]) in hexadecimal representation. (Note that the appropriate field elements d_1 and d_2 can be deduced by interpreting this representation as a series of binary coefficients $(b_i)_{i=0}^{n-1}$ of the sum $\sum_{i=0}^{n-1} b_i g^i$, where g is the generator of the corresponding finite field $\mathbb{F}_{2^n}$ and $n \in \{163, 233, 283\}$)

6.6 Arithmetical Performance of Edwards Curves

This section gives an overview of the arithmetic costs of Edwards curves and compares the fastest variants of Edwards curves to several variants of Weierstrass curves.

As in Section 2.4.5 one can get the most performance out of Edwards curves by switching to different types of projective coordinates, such as inverted Edwards curves (see [BL07c]). Table 6.2 gives an overview of the arithmetic costs of different prime field curves, where we denote the cost of each field multiplication by **M**, the cost of each field squaring by **S** and the cost of a single multiplication with a curve parameter by **D**. The parentheses next to the explicit performance costs, show the overall costs when we assume

Curve shape	Double		Add	
Weierstrass, projective	5M+6S+1D	(11M)	12M+2S	(14M)
Jacobian	1M+8S+1D	(9M)	11M+5S	(16M)
Edwards, projective	3M+4S	(7M)	10M+1S+1D	(11M)
Twisted Edwards, projective	3M+4S+1D	(7M)	10M+1S+2D	(11M)
Inverted twisted Edwards	3M+4S+2D	(7M)	9M+1S+2D	(10M)
Inverted Edwards	3M+4S+1D	(7M)	9M+1S+1D	(10M)

Table 6.2: Performance chart comparing prime field curve shapes [BL07b, BBJ$^+$08]

squarings and multiplications to be equally expensive. Note that the doubling costs of the Edwards-type curves are accomplished through explicit doubling formulas.

Remark 6.2. For a comprehensive and up-to-date overview we refer the reader to the Explicit-Formulas Database, `http://www.hyperelliptic.org/EFD`.

Chapter 7

Conclusions

In this thesis, we focused on elliptic curves in cryptography, in particular on the latest appliances of elliptic curves with regard to hash functions and on arithmetical improvements achieved through alternative curve shapes.

Part I of this thesis was devoted to the preliminaries. It focused on a comprehensive introduction to elliptic curves and hash functions as well as on the appliances of elliptic curves in cryptography.

The remainder of this thesis was split into the two core parts. Part II was dedicated to the topic of elliptic curve hashing, in which we investigated an elliptic curve hash algorithm called ECOH. ECOH participated in NIST's SHA-3 competition and hence we tried to figure out new attacks against it. Our attacks against a simplified version of ECOH were mainly based on algebraic ideas involving torsion points of ECOH's elliptic curve group. Unfortunately, Niels Ferguson and Michael A. Halcrow came up with a powerful attack against ECOH, while we were in a very early stage of our research, which rendered our efforts void [HF09]. Hence, we abandoned our attacks against ECOH and developed a memoryless version of their man-in-the-middle attack. The results of this attack are listed in Section 5.5.3.

Part III focused on a new curve shape called Edwards curves. Since elliptic curves have become an indispensable part of contemporary cryptography, they are in widespread use and hence arithmetical simplifications and speedups are of great significance. Arithmetical speedups are especially beneficial in environments with limited resources and in the context of cryptographic protocols in order to decrease the workload of servers, whereas arithmetical simplifications reduce the space required for hardware implementations and lower the vulnerability to side side-channel attacks. In the past few years much research has gone into finding new curve types according to these criteria. Edwards curves resolve all those issues as they provide a complete and amazingly fast addition law. In Chapter 6 we did not only review the theory of Edwards curves, but also deal with extensions of this theory, such as twisted Edwards curves and binary Edwards curves. The former allows us to cover more Weierstrass curves, whereas the latter describes a similar shape for binary fields. Finally, we considered to what extent standardized curves can be brought to Edwards form. However, we must conclude that only two already outdated curves over prime fields, which are part of the SEC2 standard [Sta00], have birationally equivalent twisted Edwards curves. In contrast to this, every binary Weierstrass curve can be transformed to Edwards shaped. Since in cryptographic appliances cofactors h lying in the range $1 \leq h \leq 4$ are allowed, we suggest that future curve standards should include more curves having cofactor $h \in \{2, 4\}$. This simple measure would widen the range of

standardized curves convertible to (twisted) Edwards shape.

Appendix A

Definitions

A.1 Abbreviations

DLP	Discrete Logarithm Problem
DSA	Digital Signature Algorithm
ECC	Elliptic Curve Cryptography
ECDL	Elliptic Curve Discrete Logarithm
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECM	Elliptic Curve Method
ECOH	Elliptic Curve Only Hash
ECOH^2	Elliptic Curve Only Hash 2
IFP	Integer Factorization Problem
MuHASH	Multiplicative Hash
MOV	Menezes-Okamoto-Vanstone algorithm
RSA	Rivest Shamir Adleman Cryptosystem
SSSA	Smart-Semaev-Satoh-Araki attack

A.2 Used Symbols

$\exists x$	existential quantifier
$\forall x$	universal quantifier
0^j	bit string consisting of j zero bits, e.g. $0^3 = 000$
`0x`	prefix for numbers written in hexadecimal notation
1	finite field element/integer/bit string of a single one bit
$\{0,1\}^n$	the set of all bitstrings of length n
$\{0,1\}^*$	the set of all bitstrings of arbitrary length
$\|a\|$	absolute value of the real value a
$\lfloor a \rfloor$	largest integer not exceeding a, e.g. $\lfloor 1.2 \rfloor = 1$
$\lceil a \rceil$	smallest integer not less than a, e.g. $\lceil 1.2 \rceil = 2$
$\sqrt{a}$	Square root of real value/finite field element a
$a_1, \ldots, a_6$	Weierstrass coefficients
$a \parallel b$	concatenation of two (bit) strings/vectors
$a \| b$	integer a divides b, e.g. $2 \| 4$
$a + b$	integer/group/finite field/polynomial addition

$a \cdot b$	integer/group/finite field/polynomial multiplication
a^b	integer/finite field/polynomial exponentiation
$a \bmod m$	remainder of a when divided by m
$a \oplus b$	exclusive-or (XOR) of two vectors a and b
$\left(\frac{a}{p}\right)$	Legendre symbol
$\mathbb{C}$	field of complex numbers
C_f	algebraic curve defined through function f
$C_f(K)$	set of K-rational points of curve C_f
$char(K)$	characteristic of field K
$\chi(x)$	generalized Legendre symbol
$\Delta(E)$	discriminant of curve E
E	elliptic curve in affine/projective Weierstrass form
$E^{E,c,d}$	elliptic curve in Edwards form with coefficients c and d
$E^{E,d}$	elliptic curve in Edwards form with coefficient d and $c = 1$
E^{E_B,d_1,d_2}	elliptic curve in binary Edwards form with coefficients d_1 and d_2
$E^{E_T,a,d}$	elliptic curve in twisted Edwards form with coefficients a and d
$E(K)$	set of points of curve E over field K plus the point at infinity
E/K	elliptic curve E defined over field K
$E^{M,a,b}$	elliptic curve in Montgomery form with coefficients a and b
$\mathbb{E}(X)$	expected value of some random variable X
$f \circ g$	composition of functions f and g, i.e. $f \circ g(x) = f(g(x))$
$\mathbb{F}_q$	finite field with q elements
$\frac{\partial f}{\partial X}$	formal derivation of polynomial f with respect to X
$\langle g \rangle$	group generated by some generator g
$G[m]$	torsion subgroup of group G of order m
$j(E)$	j-invariant of curve E
K	a field variable
K^*	multiplicative group of field K
$\overline{K}$	algebraic closure of field K
$k!$	factorial of (non-negative) integer k
K^n	n-dimensional (affine) vector space over field K
$K_1 \simeq K_2$	group/ring/field K_1 isomorphic to group/ring/field K_2
$K[\alpha], K(\alpha)$	extension field of field K obtained through adjunction element α
$K(X_1, \ldots, X_n)$	field of rational functions in n variables over K
$K[X_1, \ldots, X_n]$	ring of polynomials in n variables over K
$K(C_f)$	function field of the algebraic curve C_f
$K[C_f]$	coordinate ring of the algebraic curve C_f
$len(M)$	bit length of some message M
L/K	field extension of field K
$[L : K]$	degree of field extension
$\log(a)$	natural logarithm of $a \in \mathbb{R}$
$\log_2(a)$	base 2 logarithm of $a \in \mathbb{R}$
$\log_b(a)$	discrete logarithm of a with respect to generator b
$\mu_n(K)$	set of n^{th}-roots of unity of field K
$\mathbb{N}$	set of natural numbers
$\binom{n}{k}$	binomial coefficient, i.e. $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
$n^{\underline{k}}$	falling factorial, i.e. $n^{\underline{k}} = n(n-1)\ldots(n-k+1)$

$\mathcal{O}$	point at infinity on an elliptic curve
$O(\cdot)$	big O-notation
$ord(G)$	order of group G
$ord(x)$	order of group element x
$P(\cdot)$	probability measure
$\mathbb{P}$	set of all primes
$\mathbb{P}^n$	n-dimensional projective space
$\mathbb{Q}$	field of rational numbers
$\mathbb{R}$	field of real numbers
R^*	group of units of ring R
$\|S\|$	cardinality of set S
$Tr(\cdot)$	the trace function
x_P, y_P	x-coordinate and y-coordinate of some point P, respectively
$(x_1, \ldots, x_n)$	n-dimensional affine vector
$(x_1 : \ldots : x_n)$	n-dimensional projective vector
$\mathbb{Z}$	ring of integers
$\mathbb{Z}_m$	ring of integers modulo m, e.g. $\mathbb{Z}_3 = \{0, 1, 2\}$

Appendix B

Source Code

```
# Implements a simplified version of ECOH
#
# The main differences are that
#     - it is only able to create preHash values
#     - it expects message lengths to be a multiple of blen
#
class ECOH:

    # Constructor
    #
    # Parameters:
    #    E ... elliptic curve
    #    F ... underlying finite field
    #    blen ... block bitlength
    #    clen ... counter bitlength
    #    ilen ... index bitlength
    #
    # Note that out of convenience len(msg_blocks) is assumed to be a
    # multiple of blen
    #
    def __init__(self, E, blen, clen, ilen):
        self.E = E
        self.F = E.base_field()
        self.V = self.F.vector_space()
        self.m = self.V.dimension()
        # the ybit is the topmost bit of the datablock
        self.ybit = self.m - 1
        self.clen = clen
        self.ilen = ilen
        self.blen = blen
        # the padding block has the value 1||0^(blen - 1)
        self.padding_block = 1 << (blen - 1)

    # Computes the prehash of a message
    #
    # Parameters:
    #    msg_blocks ... message blocks
    # Returns:
    #    point on self.E
    #
    def getPreHash(self, msg_blocks):
        Q = self.E([0, 1, 0])
        length = len(msg_blocks)
        xor = 0

        # create points corresponding to the message blocks
        for i in range(length):
            Q += self.getPoint(msg_blocks[i], i)
            xor = xor.__xor__(msg_blocks[i])
```

```
50
51          # add the padding point
52          Q += self.getPoint(self.padding_block, length)
53          # add the XOR-point
54          Q += self.getPoint(xor, length * self.blen)
55
56          return Q
57
58      # Helper method that converts an integer into a vector representing an element
59      # of self.F.vector_space() based on its bit-representation
60      #
61      # Parameters:
62      #     x ... integer representing an element of self.F
63      # Returns:
64      #     element of self.F.vector_space()
65      #
66      def intToVector(self, x):
67          lx = x.bits()
68          while len(lx) < self.m:
69              lx.append(0)
70          return self.V(lx)
71
72      # Helper method that converts an integer into an element of self.F
73      # based on its bit-representation
74      #
75      # Parameters:
76      #     x ... integer representing an element of self.F
77      # Returns:
78      #     element of self.F
79      #
80      def intToFieldElem(self, x):
81          return self.F(self.intToVector(x))
82
83      # Helper method that determines whether an integers corresponds
84      # to a valid x-coordinate on self.E based on its bit-representation
85      #
86      # Parameters:
87      #     x ... integer
88      # Returns:
89      #     boolean
90      #
91      def isValidXCoord(self, x):
92          return self.E.is_x_coord(self.intToVector(x))
93
94      # Helper method that converts a field element to its integer representation
95      #
96      # Parameters:
97      #     x ... field element
98      # Returns:
99      #     integer
100     #
101     def fieldElemToInt(self, x):
102         v = x._vector_()
103         return sum([v[i].n().integer_part() << i for i in range(len(v))])
104
105     # Decompresses a point from a message block and an index value
106     #
107     # Parameters:
108     #     msg_block ... message block integer value
109     #     index ... index counter of this block
110     # Returns:
111     #     point on self.E
112     #
113     def getPoint(self, msg_block, index):
114         x = (((msg_block << self.ilen) + index) << self.clen) + 1
115
116         # find a valid x-coordinate
117         while self.isValidXCoord(x) == False:
```

```
118             x += 1
119
120         # get the ybit's value
121         b = (x >> self.ybit) & 0x1
122         x0 = self.intToFieldElem(x)
123         # get all the valid points
124         candidate_points = self.E.lift_x(x0, True)
125
126         # determine appropriate point according to the ybit
127         if (candidate_points[0][1] * x0^(-1))._vector_()[0] == b:
128             P = candidate_points[0]
129         else:
130             P = candidate_points[1]
131         return P
132
133 # Implements a memoryless version of the Ferguson-Halcrow attack
134 #
135 class Attack:
136
137     # Constructor
138     #
139     # Parameters:
140     #     ecoh ... ECOH instance
141     #     msg_blocks ... integer values for creation of prehash point
142     #
143     # Note that out of convenience len(msg_blocks) is assumed to be a
144     # multiple of blen
145     #
146     def __init__(self, ecoh, msg_blocks):
147         self.ecoh = ecoh
148         # bitmask for fetching blen bits
149         self.mask = (1 << ecoh.blen) - 1
150         # creating the prehash Q
151         self.Q = ecoh.getPreHash(msg_blocks)
152         # creating point X1, which is fixed
153         self.X1 = ecoh.getPoint(ecoh.padding_block, 6)
154         # creating point X2, which is also fixed
155         self.X2 = ecoh.getPoint(0, 96)
156         # small precomputation to save time
157         self.QX = self.Q - (self.X1 + self.X2)
158
159     # Helper method that determines the computation branch for some message
160     #
161     # Parameters:
162     #     msg_blocks ... array of message block integer values
163     # Returns:
164     #     integer
165     #
166     def chooseBranch(self, msg_blocks):
167         result = 0
168         for i in range(len(msg_blocks)):
169             result += Mod(msg_blocks[i], 2)
170         return result
171
172     # Helper method that computes a single step of the attack returning the
173     # resulting point, new message blocks and the branch used
174     #
175     # Parameters:
176     #     msg_blocks ... message blocks as integer values
177     # Returns:
178     #     (point on self.E, new message blocks, used branch)
179     #
180     def computeStep(self, msg_blocks):
181         # add XOR-block
182         mod_msg_blocks = [msg_blocks[0], msg_blocks[1], msg_blocks[0].__xor__(
                msg_blocks[1])]
183         # choose the computation branch
184         branch = self.chooseBranch(msg_blocks)
```

```
185
186         if branch == 0:
187             P0 = self.ecoh.getPoint(mod_msg_blocks[0], 0)
188             P1 = self.ecoh.getPoint(mod_msg_blocks[1], 1)
189             P2 = self.ecoh.getPoint(mod_msg_blocks[2], 2)
190
191             P = P0 + P1 + P2
192         else:
193             P3 = self.ecoh.getPoint(mod_msg_blocks[0], 3)
194             P4 = self.ecoh.getPoint(mod_msg_blocks[1], 4)
195             P5 = self.ecoh.getPoint(mod_msg_blocks[2], 5)
196             P = self.QX - (P3 + P4 + P5)
197
198         Px = self.ecoh.fieldElemToInt(P[0])
199         Py = self.ecoh.fieldElemToInt(P[1])
200
201         return (P, [Px & self.mask, Py & self.mask], branch)
202
203     # Implementation of Floyd's cycle finding algorithm
204     #
205     # Parameters:
206     #    msg_blocks ... message blocks as integer values
207     # Returns:
208     #    if success: second preimage
209     #    otherwise: False
210     #
211     def run(self, msg_blocks):
212         tortoise_msg_blocks = msg_blocks
213         hare_msg_blocks = msg_blocks
214         tortoise_branch = 0
215         hare_branch = 0
216
217         while True:
218             (tortoise_P, tortoise_msg_blocks, tortoise_branch) = self.computeStep(
                    tortoise_msg_blocks)
219
220             (hare_P, hare_msg_blocks, hare_branch) = self.computeStep(
                    hare_msg_blocks)
221             (hare_P, hare_msg_blocks, hare_branch) = self.computeStep(
                    hare_msg_blocks)
222
223             if (tortoise_P == hare_P):
224                 print 'MATCH found'
225                 break
226
227         tortoise_msg_blocks, hare_msg_blocks = msg_blocks, tortoise_msg_blocks
228
229         while True:
230             # store message blocks of current round
231             tortoise_old_msg_blocks, hare_old_msg_blocks = tortoise_msg_blocks,
                    hare_msg_blocks
232
233             (tortoise_P, tortoise_msg_blocks, tortoise_branch) = self.computeStep(
                    tortoise_msg_blocks)
234             (hare_P, hare_msg_blocks, hare_branch) = self.computeStep(
                    hare_msg_blocks)
235
236             if tortoise_P == hare_P:
237                 if (tortoise_branch == hare_branch):
238                     print 'undesirable MATCH found - terminating!\n'
239                     print 'restart with different message!'
240                     return False
241                 else:
242                     return (tortoise_old_msg_blocks, hare_old_msg_blocks)
```

Listing B.1: Memoryless Implementation of the Ferguson-Halcrow Attack in Sage

Bibliography

[ADH94] Leonard M. Adleman, Jonathan DeMarrais, and Ming-Deh A. Huang. A subexponential algorithm for discrete logarithms over the rational subgroup of the jacobians of large genus hyperelliptic curves over finite fields. In *ANTS-I: Proceedings of the First International Symposium on Algorithmic Number Theory*, pages 28–40, London, UK, 1994. Springer-Verlag.

[ANS98] ANSI. *ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)*. American National Standards Institute, 1430 Broadway, New York, NY 10018, USA, 1998.

[BACS08] Daniel R. L. Brown, Adrian Antipa, Matt Campagna, and Rene Struik. ECOH: the elliptic curve only hash. Submission to NIST, October 2008. `http://ehash.iaik.tugraz.at/uploads/a/a5/Ecoh.pdf`.

[BB08] Shi Bai and Richard P. Brent. On the efficiency of pollard's rho method for discrete logarithms. In *CATS '08: Proceedings of the fourteenth symposium on Computing: the Australasian theory*, pages 125–131, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[BBJ+08] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Progress in cryptology - africacrypt 2008, first international conference on cryptology in africa, casablanca, morocco, june 11-14, 2008. proceedings. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 389–405. Springer, 2008.

[Ber09] Daniel J. Bernstein. Batch binary edwards. In *CRYPTO '09: Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, pages 317–336, Berlin, Heidelberg, 2009. Springer-Verlag.

[Bir09a] Peter Birkner. AK Kryptographie. Lecture Notes, 2009.

[Bir09b] Peter Birkner. *Efficient arithmetic on low-genus curves*. PhD thesis, Technische Universiteit Eindhoven, 2009. `http://alexandria.tue.nl/extra2/200910363.pdf`.

[BK98] R. Balasubramanian and Neal Koblitz. The improbability that an elliptic curve has subexponential discrete log problem under the Menezes-Okamoto-Vanstone algorithm. *Journal of Cryptology*, 11:141–145, 1998.

[BL07a] Daniel J. Bernstein and Tanja Lange. Analysis and optimization of elliptic-curve single-scalar multiplication. Cryptology ePrint Archive, Report 2007/455, 2007. http://eprint.iacr.org/2007/455.

[BL07b] Daniel J. Bernstein and Tanja Lange. Faster addition and doubling on elliptic curves. In *ASIACRYPT*, pages 29–50, 2007. http://dx.doi.org/10.1007/978-3-540-76900-2_3.

[BL07c] Daniel J. Bernstein and Tanja Lange. Inverted Edwards coordinates. Cryptology ePrint Archive, Report 2007/410, 2007. http://eprint.iacr.org/2007/410.

[BLF08] Daniel J. Bernstein, Tanja Lange, and Reza Rezaeian Farashahi. Binary Edwards curves. Cryptology ePrint Archive, Report 2008/171, 2008. http://eprint.iacr.org/2008/171.

[BM97] Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In W. Fumy, editor, *Advances in Cryptology - EUROCRYPT '97*, number 1233 in Lecture Notes in Computer Science, pages 163–192. Springer-Verlag, 1997. Full version: http://eprint.iacr.org/1997/001.

[Bre80] Richard P. Brent. An improved Monte Carlo factorization algorithm. *Nordisk Tidskrift for Informationsbehandling (BIT)*, 20(2):176–184, 1980.

[Bro08a] Daniel R. L. Brown. Elliptic curve hash (and sign), ECOH (and the 1-up problem for ECDSA). Slides presented at ECC2008, September 2008.

[Bro08b] Daniel R. L. Brown. The encrypted elliptic curve hash. Cryptology ePrint Archive, Report 2008/012, 2008. http://eprint.iacr.org/2008/012.

[Bro09] Daniel R. L. Brown. ECOH2, June 2009.

[BSS99] Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart. *Elliptic curves in cryptography*. Cambridge University Press, New York, NY, USA, 1999.

[CF05] Henri Cohen and Gerhard Frey, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. CRC Press, 2005.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[Duq04] Sylvain Duquesne. Montgomery scalar multiplication for genus 2 curves. In *ANTS-VI: Proceedings of the 6th International Symposium on Algorithmic Number Theory*, volume 3076 of *Lecture Notes in Computer Science*, pages 153–168, Burlington, VT, USA, June 2004. Springer Berlin / Heidelberg.

[Edw07] Harold M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44(3):393–422, July 2007. http://www.ams.org/bull/2007-44-03/S0273-0979-07-01153-6/home.html.

[EJ01] Donald E. Eastlake and Paul E. Jones. Us secure hash algorithm 1 (sha1). http://www.ietf.org/rfc/rfc3174.txt?number=3174, September 2001. IETF RFC 3174.

[Eng99] Andreas Enge. *Elliptic curves and their applications to cryptography: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[Flo67] Robert W. Floyd. Nondeterministic algorithms. *J. ACM*, 14(4):636–644, 1967.

[FMR99] Gerhard Frey, Michael Mueller, and Hans-Georg Rueck. The tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Transactions on Information Theory*, 45(5):1717–1719, 1999.

[FR94] Gerhard Frey and Hans-Georg Rueck. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62(206):865–874, 1994.

[Fri05] Sophie Frisch. Algebra. Lecture notes, October 2005.

[Har60] Bernard Harris. Probability distribution related to random mappings. *The Annals of Mathematical Statistics*, 31(4):1045–1062, 1960.

[HF09] Michael A. Halcrow and Niels Ferguson. A second pre-image attack against elliptic curve only hash (ECOH). Cryptology ePrint Archive, Report 2009/168, April 2009. `http://eprint.iacr.org/2009/168`.

[HMV04] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004.

[HPS08] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *An Introduction to Mathematical Cryptography*. Springer Publishing Company, Incorporated, 2008.

[HWCD08] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted edwards curves revisited. Cryptology ePrint Archive, Report 2008/522, 2008. `http://eprint.iacr.org/2008/522`.

[Ica09] Thomas Icart. How to hash into elliptic curves. Cryptology ePrint Archive, Report 2009/226, 2009. `http://eprint.iacr.org/2009/226`.

[Int05] Internet Engineering Task Force. *ECC Brainpool Standard Curves and Curve Generation*, October 2005. `http://www.ecc-brainpool.org/download/Domain-parameters.pdf`.

[Jou00] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. In *ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory*, pages 385–394, London, UK, 2000. Springer-Verlag.

[KKM08] Ann H. Koblitz, Neal Koblitz, and Alfred J. Menezes. Elliptic curve cryptography: The serpentine course of a paradigm shift. Cryptology ePrint Archive, Report 2008/390, 2008. `http://eprint.iacr.org/2008/390`.

[KMW07] Anton Kargl, Bernd Meyer, and Susanne Wetzel. On the performance of provably secure hashing with elliptic curves. *International Journal of Computer Science and Network Security*, 7(10):1–7, October 2007. `http://paper.ijcsns.org/07_book/200710/20071001.pdf`.

[Knu69] Donald E. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd Edition) (Art of Computer Programming Volume 2)*. Addison-Wesley Professional, first edition, 1969.

[Knu97] Donald E. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd Edition) (Art of Computer Programming Volume 2)*. Addison-Wesley Professional, third edition, November 1997.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. In *Mathematics of Computation*, number 48, pages 203–209, 1987.

[Kob89] Neal Koblitz. Hyperelliptic cryptosystems. *Journal of Cryptology*, 1(3):139–150, 1989. `http://dx.doi.org/10.1007/BF02252872`.

[Kop09] Clemens Koppensteiner. Mathematical foundations of elliptic curve cryptography. Master's thesis, Vienna University of Technology, May 2009.

[LD99] Julio López and Ricardo Dahab. Fast multiplication on elliptic curves over GF(2^m) without precomputation. In *CHES '99: Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, pages 316–327, London, UK, 1999. Springer-Verlag.

[Len87] Hendrik W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987.

[Let07] Guenter H. A. Lettl. Algebraische Kurven. Lecture notes, October 2007.

[LV99] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14:255–293, 1999.

[Mil85] Victor S. Miller. Use of elliptic curves in cryptography. In *CRYPTO 85*, 1985.

[Mil04] Victor S. Miller. The Weil pairing and its efficient calculation. *Journal of Cryptology*, 17:235–261, 2004.

[Mon87] Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48:243–264, 1987.

[MOV91] Alfred J. Menezes, Tatsuaki Okamoto, and Scott Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 80–89, New York, NY, USA, 1991. ACM. `http://doi.acm.org/10.1145/103418.103434`.

[MvOV01] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001. `http://www.cacr.math.uwaterloo.ca/hac/`.

[Nat00] National Institute of Standards and Technology. *FIPS PUB 186-3: Digital Signature Standard (DSS)*. National Institute for Standards and Technology, Gaithersburg, MD, USA, January 2000. `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf`.

[Niv04] Gabriel Nivasch. Cycle detection using a stack. *Information Processing Letters*, 90(3):135 – 140, 2004.

[OKS00] Katsuyuki Okeya, Hiroyuki Kurumatani, and Kouichi Sakurai. Elliptic curves with the montgomery-form and their cryptographic applications. In *PKC '00: Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptography*, pages 238–257, London, UK, 2000. Springer-Verlag.

[OL08] Elisabeth Oswald and Mario Lamberger. Elliptic curve cryptography, an introduction to ECC and its applications in practice. Lecture notes, 2008.

[Pol75] John M. Pollard. A monte carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, 1975.

[Pra07] Norbert Pramstaller. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Graz University of Technology, October 2007.

[QD90] Jean-Jacques Quisquater and Jean-Paul Delescaille. How easy is collision search. new results and applications to DES. In *CRYPTO '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, pages 408–413, London, UK, 1990. Springer-Verlag.

[Riv92] Ronald L. Rivest. The md5 message-digest algorithm. `http://www.ietf.org/rfc/rfc1321.txt?number=1321`, April 1992. IETF RFC 1321.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. `http://doi.acm.org/10.1145/359340.359342`.

[Rue99] Hans-Georg Rueck. On the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 68(226):805–806, 1999.

[SA98] T. Satoh and K. Araki. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. *Commentarii Math. Univ. St. Pauli*, 1998.

[Saa04] Markku-Juhani O. Saarinen. Lecture 4: Hashes and message digests. Lecture Notes, February 2004. `www.tcs.hut.fi/Studies/T-79.159/2004/slides/L4.pdf`.

[Sch85] Rene Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of Computation*, 44(170):483–494, 1985.

[Sem98] Igor A. Semaev. Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p. *Mathematics of Computation*, 67:353–356, 1998.

[Ser98] Gadiel Seroussi. Table of low-weight binary irreducible polynomials. Technical Report HPL–98–135, Hewlett Packard Computer Systems Laboratories, August 1998. `http://www.hpl.hp.com/techreports/98/HPL-98-135.html`.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computation*, 26(5):1484–1509, 1997.

[Sil92] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, second edition, 1992.

[Sma99] Nigel P. Smart. The discrete logarithm problem on elliptic curves of trace one. *Journal of Cryptology*, 12:193–196, 1999.

[Sma09] Nigel P. Smart. Cryptography. an introduction. E-book, January 2009. `http://www.cs.bris.ac.uk/~nigel/Crypto_Book/`.

[SOK00] Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. Cryptosystems based on pairings. In *Proceedings of the 2000 Symposium on Cryptography and Information Security (SCIS '00)*, Okinawa, Japan, 2000.

[Sta00] Standards for Efficient Cryptography Group (SECG). *SEC 2: Recommended Elliptic Curve Domain Parameters*, September 2000. `http://www.secg.org/download/aid-386/sec2_final.pdf`.

[Tat66] John Tate. Endomorphisms of abelian varieties over finite fields. *Invent. Math.*, 2:134–144, 1966.

[TF03] Robert Tichy and Clemens Fuchs. Mathematische Grundlagen der Kryptographie. Lecture Notes, 2003.

[vOW94] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In *CCS '94: Proceedings of the 2nd ACM Conference on Computer and communications security*, pages 210–218, New York, NY, USA, 1994. ACM.

[Wag02] David Wagner. A generalized birthday problem. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 288–303, London, UK, 2002. Springer-Verlag.

[Was08] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. Discrete Mathematics and Its Applications. Chapman & Hall/CRC, second edition, April 2008.