Performance Evaluation of Optimal Ate Pairing on Low-Cost Single Microprocessor Platform

Thesis, Network security

Mikko Pesonen
September 2020

University of Turku
Department of Future Technologies
Supervisors: Dr.Sc., Adj. Prof. Ethiopia Nigussie, M.Sc. Marko Winblad

University of Turku
Department of Future Technologies

PESONEN MIKKO: Performance Evaluation of Optimal Ate Pairing on Low-Cost Single Microprocessor Platform
Thesis, 56 p., 0 app. p.
Network Security
September 2020

The framework of low-cost interconnected devices forms a new kind of cryptographic environment with diverse requirements. Due to the minimal resource capacity of the devices, light-weight cryptographic algorithms are favored. Many applications of IoT work autonomously and process sensible data, which emphasizes security needs, and might also cause a need for specific security measures.

A bilinear pairing is a mapping based on groups formed by elliptic curves over extension fields. The pairings are the key-enabler for versatile cryptosystems, such as certificateless signatures and searchable encryption. However, they have a major computational overhead, which coincides with the requirements of the low-cost devices. Nonetheless, the bilinear pairings are the only known approach for many cryptographic protocols so their feasibility should certainly be studied, as they might turn out to be necessary for some future IoT solutions. Promising results already exist for high-frequency CPU:s and platforms with hardware extensions.

In this work, we study the feasibility of computing the optimal ate pairing over the BN254 curve, on a 64 MHz Cortex-M33 based platform by utilizing an optimized open-source library. The project is carried out for the company Nordic Semiconductor. As a result, the pairing was effectively computed in under $26 \cdot 10^6$ cycles, or in 410 ms.

The resulting pairing enables a limited usage of pairing-based cryptography, with a capacity of at most few cryptographic operations, such as ID-based key verifications per second. Referring to other relevant works, a competent pairing application would require either a high-frequency — and thus high consuming — microprocessor, or a customized FPGA. Moreover, it is noted that the research in efficient pairing-based cryptography is constantly taking steps forward in every front-line: efficient algorithms, protocols, and hardware-solutions.

Keywords: pairing-based cryptography, embedded platform, optimal ate pairing

Contents

List of Algorithms

List of Figures

Chapter 1

Introduction

The emerging field of interconnected, smart devices has posed new challenges in the field of information security. The embedded devices are usually very small-scale and endowed with constrained resources, which limits the usage of security algorithms. Low-weight protocols utilizing elliptic curve cryptography (ECC) are adopted as the main component for accomplishing security. The devices may handle sensitive data, and the composed applications might have special requirements, comparing to the usual computer networks.

In cryptography, the concept of *bilinear pairing* was originally brought forth in 1993, in the form of MOV-attack [27] which exploits pairings to attack the elliptic curve protocols that utilize a specific class of curves. However, after a two decades of research, it has turned out that bilinear pairings enable a novel class of cryptographic applications. These include the identity based encrypiton [11], various certificateless signature schemes, searchable encryption [26], a signcryption in wireless sensor networks [19], and many other applications with interesting properties.

A major bottleneck in the pairing implementations is posed by the massive runtime and resource requirements of the computations. Consequently, a lot of research contribution has been given to algorithmic, arithmetic and implementation issues of the pairings. Research has attained the implementations on constrained, embedded platforms. A single pairing has been computed in 3.88ms, with a 1.2 GHz Rasberry Pi3 platform [22]. Even smart card implementations have been presented [17]. However, the majority of the implementations for the ultra-low-power devices are quite impractical, taking hundreds of milliseconds to compute a single pairing.

The pairings offer unique, new kind of security mechanisms, which would be difficult, if not impossible, to construct with other known cryptographic primitives. Hence the possibility of using them in embedded platform should certainly be studied. The accelerating increase of interconnected devices and new kinds of applications will pose new challenges on information security and privacy, which may emerge a need for novel kinds of security protocols. This is where the pairing protocols might show their significance. Pairing-based cryptography has already found innovative applications in various systems, such as in Industrial Internet of Things (IIoT)[34], wearable technologies and health record systems [25] and sensor networks [19].

1.1 Thesis Objective and Research Questions

The objective of this thesis is to evaluate the performance of optimal ate pairing over the BN254 curve in resource-constrained single microprocessor embedded platform. The study gives a reference for the feasibility of pairing-based cryptography on a low-cost general-purpose embedded platform. The pairings enable novel kinds of security features in IoT and other embedded solutions, which gives a reason to study them.

The main research problem of the thesis is studying the feasibility of optimal ate pairing in a resource-constrained single microprocessor platform. The problem is divided into three research questions: 1) is the computation of optimal ate-pairing in constrained microprocessor feasible? 2) does the resulting pairing enable adopting the pairing-based protocols and related approaches? 3) how the existing implementation could be further enhanced?

1.2 Thesis Structure

This thesis is structured as follows. Chapters 2-3 form the preliminary theory part, including requisite cryptography, mathematical fundamentals — such as fields and elliptic curves, and ultimately the topic of pairing-based cryptography. Elliptic curve cryptography (ECC) is presented shortly. Chapter 4 covers the key methodology for efficient pairing computations, focusing on the methods adopted in this work. In chapter 5, our proposed implementation and results are presented, in addition to a discussion of the further improvements and the current situation of pairing-based cryptography. Chapter 6 contains the conclusions.

Chapter 2

Some Preliminary Concepts

This chapter starts by outlining the preliminary terminology and concepts behind the bilinear pairings. The first section gives a brief overview of cryptographic terminology, with emphasis on relevant themes. The second section introduces some necessary mathematical concepts that are in a key role in understanding the concept of elliptic curve cryptography, and furthermore the bilinear pairings. The third section contains the elliptic curve subject. It introduces the elliptic curves over finite fields and discusses briefly their security and usage in cryptography.

2.1 Overview of Cryptography

Cryptography is the science and study of methodology for secure communication over an insecure channel. The goal is to fulfill confidentiality, integrity, authenticationn and non-repudiation of information, or transactions. Confidentiality is fulfilled with *encryption* of the data, making it uninterpretable for unauthorized parties. Integrity is accomplished with checksums and digital signatures, which ensure that the data has not been altered by a third party. Authentication, and in some cases, non-repudiation is fulfilled with digital signatures and *Message Authentication Codes* (MACs), which ensure that certain transaction is indisputably performed by a certain party.[33]

Encryption is a process of transforming a *plain text* into apparent noise, *ciphertext*. *Decryption* is the reversal process, converting the ciphertext back into readable form. A *key* is closely related to cryptographic processes, such as message encryption and performing digital signing. The *keyspace* is the set of all possible keys. *Cryptanalysis*, in turn, is the science of breaking cryptographic systems. It can refer to finding out the secret key or reversing the cryptographic process without the required key.[33]

Cryptography is usually divided into symmetric and asymmetric algorithms, and into different cryptographic protocols. In a symmetric algorithms, one secret key is used for both encryption and decryption of data. Asymmetric algorithms rather incorporate a key pair, of which one key is public and one is kept secret. They are also known as *public-key* algorithms. Cryptographic protocols deal with the actual applications, employing both symmetric and asymmetric techniques.[33] In this document, a set of algorithms for achieving a certain security attribute is called a *cryptosystem*.

Symmetric algorithms are mainly used for data encryption and integrity verifying algorithms. Advanced Encryption Standard (AES) is one of the most common symmetric ciphers[1]. Public-key algorithms have more versatile applications. Public-key cryptography addresses two fundamental problems — secure distribution of the secret value over an insecure channel, and the creation of unforgeable digital signatures. With a key distribution protocol, the actual key for message encryption can be exchanged over an untrusted channel. Digital signatures, as mentioned, provide authentication and non-repudiation for a piece of information or action. Public-key systems are also capable of encryption and verifying integrity, but they are primarily used in key exchange and signature-generating. Common public-key algorithms are RSA, ElGamal and the elliptic curve-based systems.[33]

2.2 Security of Cryptosystems

Security of a cryptosystem is dependent on the keyspace but more importantly of the complexity of the equivalent *hard problem* that is needed to be solved in order to break the system. The cryptographic operation should be infeasible[2] to be inverted without knowledge of the secret key. Similarly, the information generated by the cryptographic process should not reveal any information about the key. A secure cryptosystem should have resilience against all known applicable methods of cryptanalysis.

The principle behind public-key cryptosystems is the *one-way* property. A function $f : X \to Y$ fulfills the one-way property if $f(x) = y$ can be efficiently computed for any $x \in X$, but solving $f^{-1}(y) = x$ is computationally infeasible. Meaning that the $x \in X$ for corresponding $f(x) = y \in Y$ cannot be found in polynomial time. In practice,

[1]A cipher is here defined as a pair of algorithms; for encryption and decryption

[2]Infeasibility is here loosely defined as "no efficient attempts or methods known to compute or solve a problem in polynomial time".

the one-way property is accomplished with a certain difficult problem, which has a *back door* — a secret piece of information for inverting the function.[33] Examples of one-way properties (and corresponding cryptosystems based on it) are the discrete logarithm problem (ElGamal), integer factorization problem (RSA) and elliptic curve discrete logarithm problem (ECC and bilinear pairings).[23]

Cryptosystems are also vulnerable to physical attacks, which take advantage of leaked information of the physical implementation of the system. By monitoring some by-product generated by the corresponding device, an adversary might gather information about the internal state or operations, and thus, piece-by-piece, construct the secret key.

As mentioned in [30], physical attacks against elliptic curve based systems are divided into *Side-Channel Analysis* (SCA) and *fault attacks*. Attacks based on SCA utilize leaked power traces, timing, and electromagnetic radiation. For instance, *Simple Power Analysis*-attack (SPA) makes use of key-dependent patterns that occur on power traces, which can be used to deduce information about the key. Calculations with dummy variables can be used to reduce the predictability of the process to some extend. However, the usage of dummy calculations poses another weakness. In fault attacks, the adversary aims to disturb the cipher to derive secrets by injecting faults during certain computation phases. Faults can be induced with glitches in the clock or with a laser beamer, for instance. An error induced during dummy calculation can reveal parts of the secret key.[30]

2.3 Some Mathematical Prerequisites

Foundations of abstract algebra provide the basis for the majority of public-key cryptosystems, as well as for ECC and pairing-based protocols. As the focus of this thesis is more on practical side, a detailed mathematical presentation and derivation of the cryptographic structures is not appropriate. However, the very essential theory is presented in a non-rigorous manner. The goal is that a reader with a minor mathematical background gets a brief overview of the necessary mathematical elements for understanding the elliptic curves and bilinear pairings adequately.

This section includes basics of groups and fields, compressed in under three pages, and some distinct notions. The main source for this section is [30] and [39] which can be referred for further information.

The notion of a group is a constitutive building block in cryptography.

Group $(G, \circ)$ is a set of elements G endowed with a binary operation $\circ$ that satisfies the following properties:

1. Group is closed under the group operation $\circ$, i.e. $\circ : G \times G \to G$.
2. Operation $\circ$ is associative: $(a \circ b) \circ c = a \circ (b \circ c) \quad \forall a, b, c \in G$.
3. Exists an unique neutral element $e \in G$ such that $a \circ e = e \circ a = a \quad \forall a \in G$.
4. Each element $a \in G$ has an inverse $a^{-1} \in G$ for which $a \circ a^{-1} = a^{-1} \circ a = e$.

If also $a \circ b = b \circ a$ for all $a, b \in G$, the group is Abelian, or *commutative*. Groups can be finite or infinite. In this document, only finite groups are considered. The size, also called *order* of a group G is denoted as $\#G$.[30]

Some common groups are $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$ (integers modulo n) under addition, and $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}$ under multiplication.

If a certain set $H \subset G$ is a group under the restriction of binary operation defined on G, then H is a *subgroup* of G, denoted as $H < G$. Every element $a \in G$ generates a subgroup of G. This subgroup is composed of elements constructed from a, applying the group operation consecutively. The size of the subgroup generated by a is denoted as $ord(a)$, which refers to the order of the element a.[30] The next theorem gives valuable information about subgroups.

Theorem 2.1 ([30])**.** *Lagrange's Theorem. Let G be a finite group. If H is a subgroup of G, then the order of H divides the order of G, i.e. $\#H | \#G$. In other words, if $g \in G$, then $ord(g) | \#G$.*

As a consequence, the groups with prime order, i.e. $\#G = p \in \mathbb{P}$, have only trivial subgroups: the group G itself and the trivial group $\{1\}$.

If a group G is generated by one element a, it is called *cyclic*, and is denoted as $\langle a \rangle = G$. Subgroups of cyclic groups are cyclic. I.e. if $G = \langle g \rangle$ and $H < G$, then $\exists k \in \mathbb{Z}$, $\langle g^k \rangle = H$.[30] This gives an intuition to the fact that the order $ord(a)$ can be thought as the smallest positive integer h for which $\underbrace{a \circ a \circ \cdots \circ a}_{h \text{ times}} = 1$, where 1 is the identity element of the group. The following theorem completely characterizes the subgroups of a (finite) cyclic group.

Theorem 2.2 ([33]). *Let $G = \langle g \rangle$ be finite cyclic group, with $\#G = n$. For every $m \in \mathbb{Z}$ such that $m|n$ exists a subgroup $H \leq G$ of order m. This subgroup is generated as $\langle g^{n/m} \rangle$. There are no other subgroups.*

Cryptosystems based on cyclic groups are usually endowed with a strong one-way property called *Discrete logarithm problem* (DLP). If $\alpha \circ \alpha \circ \cdots \circ \alpha = \beta$ (group operation $\circ$ is applied k times), the DLP is to find the corresponding k.[33] In other words, solving $\log_\alpha \beta = k$. The logarithm here is quite ambiguous as it depends of the used group operation. Usually the question of interest is to solve DLP in the multiplicative group, i.e. $\alpha^k \equiv \beta \pmod p$ for some integer k. The most efficient known method for solving DLP in $\mathbb{Z}_p^*$ is the *index calculus method* which has a sub-exponential time complexity. DLP for elliptic curves is revisited in chapter 3.1.

Definition 2.1 ([30]). *Homomorphisms. Let $(G, *)$ and $(G', \circ)$ be groups. The mapping $f : G \rightarrow G'$ is called a* homomorphism *if $f(a * b) = f(a) \circ f(b)$ for all $a, b \in G$. In other words, the mapping f takes the operations in G to the corresponding operations in G'. The* kernel *of f is $Ker(f) = \{a \in G \mid f(a) = 0\} = f^{-1}(\{0\})$ and the* image *of f is $f(G)$.*

If the mapping f is bijective, the groups G and G' are *isomorphic* — denoted as $G \simeq G'$.[30]

Arithmetic over finite fields is employed in elliptic curve cryptosystems, and consequently in bilinear pairings.

Definition 2.2 ([30]). *Field. A triplet $(F, +, \cdot)$ is a field if following properties satisfy:*

1. *$(F, +)$ is Abelian group with a neutral element 0.*
2. *$(F \setminus \{0\}, \cdot) = (F^*, \cdot)$ is Abelian group with a neutral element 1.*
3. *Distributivity law holds: $\forall a, b, c \in F : a(b + c) = (ab) + (ac)$.*

A field can be thought as a group with some additional properties. In elliptic curve cryptography and bilinear pairs, large (but finite) fields and a certain *field extensions* are involved. Next, the basic outline of these fields and their structures is given in a simplified form.

Majority of elliptic curve cryptography is based on two fields: the *prime field* and the *binary field.* A prime field, with respect to certain prime p is denoted as

$$\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = \{0, 1, 2, \ldots, p - 1\}.$$

Which is similar to the group $\mathbb{Z}_m$ defined earlier, but with $m = p \in \mathbb{P}$. The notation G/N refers to a *quotient ring.*[30]

The binary field is actually a *field extension.* Field extensions $\mathbb{F}_{p^n}$, where $p \in P$ and $n \in \mathbb{N} \setminus \{0\}$ are constructed with irreducible polynomials. Polynomial $f(x) \in K[x] = \{a_0 + a_1x + \ldots + a_nx^n \mid n \geq 0, a_i \in K\}$ is irreducible if and only if $f(x)$ is neither a constant polynomial nor a product of two polynomials of positive degree. Let n be the degree of irreducible polynomial $f(x)$, then the prime field extension is defined as

$$\mathbb{F}_{p^n} = \{a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} \mid a_i \in \mathbb{F}_p, f(x) = 0\}$$

If $p = 2$, then the prime field extension is a binary field, with coefficients from $\mathbb{F}_2 = \{0, 1\}$.[23] An element of a binary field can be interpret as a binary string. For instance, $x^{12} + x^9 + x^5 + x + 1$ corresponds to a 13 bit value `1001000100011`. Consequently, arithmetic of the binary field elements is efficiently carried out with logical operations.

Example 1. *In a prime field $\mathbb{F}_p$, the equality $(a + b)^p = a^p + b^p$ applies. This is easily proven as $(a+b)^p = a^p + b^p + \sum_{k=1}^{p-1} \binom{p}{k} a^k b^{p-k} \equiv a^p + b^p \pmod{p}$. This property simplifies exponentiation over prime fields and field extensions.*

Example 2. *Example of addition and multiplication over a prime field extension $\mathbb{F}_{2^5} = \{a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 \mid a_i \in \{0, 1\}, 1 + x + x^5 = 0\}$.*

As $x^5 + x + 1 = 0$, x^5 is reduced to $x + 1$ in computations, recalling that $-1 \equiv 1$ and $2 \equiv 0$ in $\mathbb{F}_2$. Now proceeding the computation with two elements of $\mathbb{F}_{2^5}$:

$$\begin{aligned}(x^3 + x^2 + 1) + (x^3 + x^2) &= 1 \\ (x^3 + x^2 + 1) \cdot (x^3 + x^2) &= x^6 + x^5 + x^5 + x^4 + x^3 + x^2 \\ &= x \cdot x^5 + x^4 + x^3 + x^2 \\ &= x(x + 1) + x^4 + x^3 + x^2 \\ &= x^4 + x^3 + x\end{aligned}$$

Let K and L be fields such that $K \subseteq L$. Element $\alpha \in L$ is said to be algebraic over K, if there exists a non-constant polynomial $f(x) = x^n + a_{n-1}x^{n-1} + \ldots + a_0$, with $a_0, \ldots, a_{n-1} \in K$, such that $f(\alpha) = 0$. If every element of L is algebraic over K, then L is an algebraic extension of K.[39]

Definition 2.3 ([39]). *Algebraic closure. An algebraic closure of a field K is a field $\bar{K}$, $K \subset \bar{K}$ such that*

1. *$\bar{K}$ is algebraic extension of K*

2. *Every non-constant polynomial $f(x)$ with coefficients in $\bar{K}$ has a root in $\bar{K}$. In other words: $\bar{K}$ is algebraically closed.*

Example 3. *Field $\mathbb{R}$ is not an algebraic closure, as the polynomial $x^2 + 1 = 0$ has no roots over the field of real numbers.*

2.4 Elliptic Curves over Finite Fields

The insight of using elliptic curves in cryptography was formed in the middle '80s, independently by two researchers Neal Koblitz and Victor Miller [24, 29]. After two decades of research, Elliptic Curve Cryptography (ECC) is standardized and widely used. ECC is based on the foundation that elliptic plane curves over finite fields form an Abelian group, under a certain geometric operation. This allows group-theoretic machinery to be applied, resulting in efficient and secure cryptographic protocols.

Elliptic curves provide efficient tools for key agreement, digital signatures, but also for non-cryptographic applications, such as integer factorization. Besides, groups over elliptic curves form the basis for all known bilinear pairings. The ECC protocols have many advantages compared to prior public key protocols. Whilst fulfilling the same security level as RSA, they offer more efficient implementations and smaller key sizes.

The main sources for this section are [23], [39], [30] and [7].

The Elliptic Curve concept originates from an idea of combining geometric properties of elliptic curves and calculations over finite fields. The field elements satisfying the elliptic curve-equation form an Abelian group. The elliptic curve E is defined by a *non-singular generalized Weierstrass equation* over a finite field K as

$$y^2 + a_1xy + a_2y = x^3 + a_3x^2 + a_4x + a_5,\ a_i \in K. \tag{2.1}$$

An elliptic curve is thereby defined by the set of points $(x, y) \in K^2$ satisfying the curve equation (2.1), and is denoted by $E(K)$. Weierstrass equation is the generalized version of elliptic curves. With a linear change of variables, variants of the curve (2.1) can be derived for different fields, and different applications. In case of a prime field $(p \neq 3)$, the equation has a form of

$$y^2 = x^3 + ax + b \tag{2.2}$$

where $a, b \in \mathbb{F}_p$, and $\Delta = -4a^3 - 27b^2 \neq 0$.[23]

In case of binary field, the equation is:

$$y^2 + xy = x^3 + ax^2 + b \tag{2.3}$$

where $a, b \in \mathbb{F}_{2^m}$, and it is further required that $\Delta = b \neq 0$. This curve is labeled as the *binary curve.*[23]

The curves above are of *Short-Weierstrass* form. The discriminant property $\Delta \neq 0$ ensures that the roots of the elliptic curve are distinct, i.e. the curve does not contain "bad points" such as cusps or self-intersections. By the definition, this *non-singularity* is required for a curve to be an elliptic curve.[33]

2.4.1 The Group Structure

An elliptic curve is defined as the set of coordinates (elements of K) on the plane curve E as follows:

$$E(K) = \{(x, y) \in E : x, y \in K\} \cup \{\mathcal{O}\} \tag{2.4}$$

As will be shown, $E(K)$ forms a group under the defined geometric operation. The point $\mathcal{O}$ is called a *point at infinity* — a point to infinitely far on the y-axis. It is used as the neutral element of the group.[23]

The additive group operation $\oplus$ is called *chord-tangent process.* The process is best visualized with a picture (see fig. 2.1). A straight line through P and Q is formed, as $y = \lambda x + \nu$, and its intersection with the curve E is calculated. The resulting point $P \oplus Q$ locates on the opposite intersection point of the curve.[39] It should be noted that in finite fields, the curve is a set of scattered points, as depicted in figure (2.2).

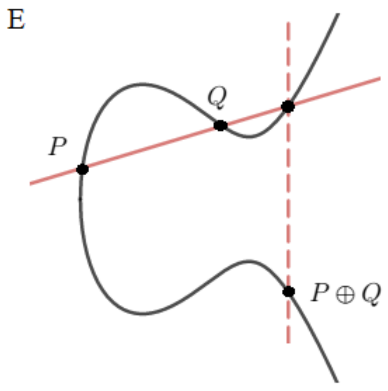

Figure 2.1: Chord-tangent process

Next, the formation of the group operation will be studied more in detail, based on [39]. The group operations are presented for affine coordinates. First let $P = (x_1, y_1) \neq \mathcal{O}$ and $Q = (x_2, y_2) \neq \mathcal{O}$ be two points on the curve $y^2 = x^3 + ax + b$, over a prime field $(p > 3)$. Let us now proceed with the aforementioned geometrical steps in terms of computations for $P \oplus Q = (x_3, y_3)$.

First, the chord $y = \lambda x + \nu$, connecting P to Q is computed. The slope of the line y

is calculated as

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & \text{if } P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1} & \text{if } P = Q \end{cases} \tag{2.5}$$

and $\nu = y_1 - \lambda x_1$. Note that if $P = Q$, λ represents the tangent of the curve at point $P = (x_1, y_1)$, and is thus derived by implicit differentiation of the curve equation.

Now substituting the equation of the chord to the eq. 2.2 gives $(\lambda + \nu x)^2 = x^3 + ax + b$. As x_1 and x_2 are known solutions of $x^3 + ax + b - (\lambda + \nu x)^2 = 0$, the third solution x_3 is easily found: $x_3 = \lambda^2 - x_1 - x_2$. Therefore $y_3 = -\lambda x_3 - \nu$, and finally

$$P \oplus Q = (x_3, y_3) = (\lambda^2 - x_1 - x_2, \lambda(x_1 - x_3) - y_1). \tag{2.6}$$

It can be shown that group rules are satisfied under this group operation: $P \oplus (Q \oplus R) = (P \oplus Q) \oplus R$ and $P \oplus Q = Q \oplus P$ for all $P, Q, R \in E$ (*associativity* and *commutativity* of the group); $P \oplus \mathcal{O} = \mathcal{O} \oplus P = P$ and $P \oplus (-P) = (-P) \oplus P = \mathcal{O}$ for any $P \in E$ (identity and inverse elements of a group). Therefore, $(E, \oplus)$ indeed forms an Abelian group. For rigorous proofs for group rules, one may refer to [39]. The obtained formulas apply in prime fields as-is. For other types of fields and equations, the operations are derived in a similar fashion. In example 4, the group operation on a curve over $\mathbb{F}_{19}$ is illustrated.

The scalar multiplication kP on Elliptic curve E is defined as k repeated additions of a point $P \in E$ to itself: $kP = P \oplus P \oplus \ldots \oplus P$. Order of an element P is the smallest positive integer n for which $nP = \mathcal{O}$.[39]

Theorem 2.3 ([39])**.** *(Hasse's Theorem). Let E be an elliptic curve over the field K with q elements. Then the size of an elliptic curve E has the following bounds:*

$$(\sqrt{q} - 1)^2 \leq \#E(K) \leq (\sqrt{q} + 1)^2.$$

Inequality can be written as $|\#E(K) - (q + 1)| \leq 2\sqrt{q}$ where $\#E(K) - (q + 1)$ is also known as the *trace of frobenius*. It is worth noting that Hasse's theorem only gives bounds for the size of E — the exact number of points is rather difficult to determine. One of the most efficient methods for point computation is the Schoof's algorithm [35], which has a complexity of $\mathcal{O}(\log^8 q)$.

Definition 2.4 ([39])**.** *n-torsion points. Let $E(K)$ be an elliptic curve and n be a positive integer. The set of n-torsion points of E is defined as:*

$$E[n] = \{P \in E(\bar{K}) \mid nP = \mathcal{O}\},$$

where $\bar{K}$ is the algebraic closure of K. More precisely, the set $E[n]$ contains points of elliptic curve that have a finite order.

Torsion points are namely the points that have finite order. In terms of finite fields, every point is a torsion point.[3][39]

Elliptic curves $E(\mathbb{F}_q)$, where $q = p^k$ are called *supersingular*, if $E(\mathbb{F}_q)[p] = \{\mathcal{O}\}$ i.e. the curve has no points of order p. This is equivalent to $p \mid \#E(\mathbb{F}_q) - (q+1)$, i.e. the trace of Frobenius is a multiple of p. If $\#E(\mathbb{F}_q) = q$, i.e. the trace of Frobenius is 1, the elliptic curve is called *anomalous.*[39]

Supersingular and anomalous curves are considered cryptographically weak.[39] Supersingular curves are vulnerable to the MOV-attack [27] which is described briefly in chapter 4.1. Anomalous curves have low resilience against DLP over elliptic curves. Supersingular curves are, however, employed in some use cases. Note: supersingularity is not to be confused with singularity, which was defined in the section 3.1.

Example 4. *Considering an elliptic curve $E : y^2 = x^3 - 7x + 10$ over a small prime field $\mathbb{F}_{19}$. All the points $(x, y) \in \mathbb{F}_{19}^2$ satisfying E, i.e. solutions for $y^2 \equiv x^3 - 7x + 10 \pmod{19}$ are:*

$(1,2)$	$(5,9)$	$(10,16)$	$(16,17)$
$(1,17)$	$(5,10)$	$(12,1)$	$(17,4)$
$(2,2)$	$(7,0)$	$(12,18)$	$(17,15)$
$(2,17)$	$(9,7)$	$(13,8)$	$(18,4)$
$(3,4)$	$(9,12)$	$(13,11)$	$(18,15)$
$(3,15)$	$(10,3)$	$(16,2)$	$\mathcal{O}$

24 points on E are found (including the neutral element $\mathcal{O}$), which is within Hasse's bounds as $11 < (\sqrt{19} - 1)^2 \leq \#E(\mathbb{F}_{19}) \leq (\sqrt{19} + 1)^2 < 29$.

Let $P = (13, 11)$, $Q = (17, 15)$, $P, Q \in E(\mathbb{F}_{19})$. Now $P \oplus Q = (x_3, y_3)$ can be computed using the equations (2.5) and (2.6).

As $P \neq Q$, $\lambda = \frac{15-11}{17-13} = 4 \cdot 4^{-1} = 1$. Hence $x_3 = 1^2 - 13 - 17 \equiv 9 \pmod{19}$ and $y_3 = 1 \cdot (13 - 9) - 11 \equiv 12 \pmod{19}$. Thus $P \oplus Q = (9, 12)$. The group operation is visualized in the figure (2.2) below.

[3]But there might be no points in $E[n]$ for each $n \in \mathbb{F}_p$.

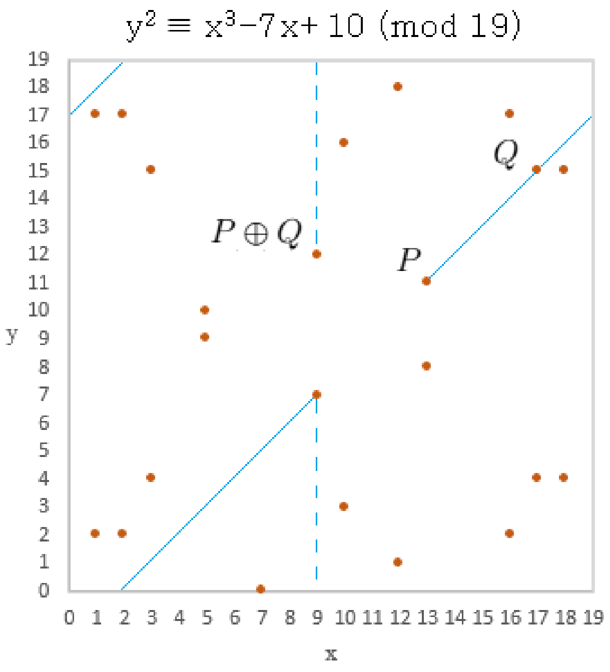

Figure 2.2: Chord-tangent process on a curve over $\mathbb{F}_{19}$

2.4.2 Elliptic Curve Discrete Logarithm Problem

As the elliptic curve cryptography is based on cyclic groups, the computation of discrete logarithm is also the underlying hard problem of the protocol. But as the group operation on elliptic curves is different, the problem of interest is appointed as Elliptic Curve DLP (ECDLP). For elliptic curves, the problem is to solve the coefficient k in scalar multiplication $kG = G \oplus G \oplus \ldots \oplus G = P$, while knowing the base point G and $P \in \langle G \rangle$.[23]

With a proper choice of the curve, the best algorithms for ECDLP are the generic DL algorithms, that work on arbitrary cyclic groups. Such methods, for example, the Pollard's rho algorithm can compute generic DLP probabilistically in at most $\mathcal{O}(\sqrt{n})$ steps.[23]

2.5 Elliptic Curve Cryptosystems

Elliptic curve protocols have remarkable advantages over the other respective public-key systems in terms of key size and performance. Considering a 3072-bit RSA key, the corresponding security level is obtained with only 256-bit ECC key.[23] Another advantage within ECC are the standardized curves and parameters; the user only needs to generate a random number s as the private key. In RSA, the security specifically relies on the generated parameters. That is, two strong primes with a size over 1000 bits must always be generated and kept secret, when deploying the cryptosystem.

In Elliptic Curve protocols, the system parameters are the used prime p for a prime field, or m in case of binary field $\mathbb{F}_{2^m}$; constants a and b defining the curve, and the generator point G, and its order r. Private key of the user is a random integer $s \in \mathbb{Z}_r$. Public key is the point $Q = sG$. If a binary field is used, the generator polynomial f is also included.[7] In this section, two common elliptic curve protocols are presented: the elliptic curve variant of Diffie-Hellman (ECDH) and elliptic curve digital signature protocol (ECDLP).

Elliptic Curve Diffie-Hellman

The Diffie-Hellman protocol allows secret key exchange over an insecure channel. During the process, certain pieces of public information are transmitted over the channel, which is then combined with at least one piece of secret information to obtain the key. A possible eavesdropper can not obtain the key, as he neither possesses the secret piece(s) of the information nor is capable to compute it from the public information. The process is also known as key-agreement.[39]

Elliptic curve version of *Diffie-Hellman* (ECDH) uses the scalar multiplication of an elliptic curve point as the operation. The particular finite field $\mathbb{F}_q$, the curve E over it, and a generator point G and its order r are known to communication parties A and B. Now the secret key is computed as follows:

1. A and B individually generate random integers $a, b \in \mathbb{Z}_r^*$, and compute $P = aG$ and $Q = bG$ respectively.
2. A sends P to B and B sends Q to A.
3. A and B compute the point $aQ = a(bG) = b(aG) = bP$ (commutativity of E is used). The shared secret key is the extracted x-coordinate of the generated point.

Now finding the $(ab)G = k$ would require the eavesdropper to solve either a or b from k, which is known as *Diffie-Hellman problem.* The problem is, according to the current knowledge, computationally equivalent to solving the DLP.[39]

Elliptic Curve Digital Signature Algorithm

Let $E(\mathbb{F}_q)$ be an elliptic curve, $G \in E$ a generator point, and $r = ord(G)$. The parameters are assumed known to communication parties. Also let $s \in \mathbb{Z}_r^*$ be the private key and $Q = sG$ public key of communication party A. Now, A signs a message as follows:

1. A computes the hash $h(m)$ of a message, with an agreed hash algorithm.
2. A selects a random, secret integer $k \in \mathbb{Z}_r^*$, computes the point $P = kG$ and extracts its x-coordinate as an integer c using a simple, pre-determined conversion function.
3. The signature is the pair (c, d) where $d \equiv k^{-1}(h + sc) \pmod r$.

Any other party B is now able to verify the signature of A for message m as follows:

1. B computes $h = h(m)$ with the same hash function as used in signing process.
2. B, knowing the public key Q, calculates the point $R = d^{-1}(hG+cQ)$ and extracts its x-coordinate as an integer c'.
3. B accepts the signature if $c' = c$.

Signature verification works, as

$$R = d^{-1}(hG + cQ) = (k^{-1}(h + sc))^{-1}(hG + scG) = k(h + sc)^{-1}(h + sc)G = kG,$$

which equals the point calculated by A in the signing process. For an attacker to counterfeit the signature of A, solving the secret key s from $Q = sG$ is required. This equals, again, solving the ECDLP.[23]

It is clear that $r \neq 0 \neq s$. Also, notable is that k is considered to be a NONCE (Number used ONCE), and must be kept secret. If corresponding k for a certain message m is revealed, the secret key can be computed as $s = c^{-1}(dk - h(m)) \pmod r$. If the same random k is used more than once with two different messages $h(m_1)$ and $h(m_2)$, it can be solved. Considering $k = d_i^{-1}(h(m_i) + sc_i)$, for signatures $i = 1, 2$ with equivalent k:s. As $k_1 = k_2$ result into $c_1 = c_2$, it follows that $k \equiv (d_1 - d_2)^{-1}(h(m_1) - h(m_2))$. Consecutively, the private key s can be solved.[23]

2.6 Secure Curve Parameters

This section summarizes some general key-points of suitable elliptic curves for cryptography. The current standards and recommendations for elliptic curve cryptography are also studied, mainly based on the criteria provided by NIST 800-186 [7]. Certicom standards [13] are also used for reference.

The used elliptic curve $E(K)$ should have a significantly large subgroup of prime order, that is, $\#E = n \cdot h$ where n is a large prime factor and h is a relatively small co-factor. If the co-factor h is too large, the curve is exposed to the *Pohlig-Hellman* attack, which takes advantage on small prime factors of the group order.[39] NIST suggests a co-factor of $h \leq 2^{10}$ to be used. If the prime factor n is not large enough, the curve is vulnerable to Pollard's Rho attack for solving ECDLP in the prime subgroup of E.

Moreover, a secure elliptic curve should not be supersingular nor anomalous. Non-supersingularity is achieved with a large *embedding degree*, which is explained in the section 4.1. As mentioned, supersingular curves are vulnerable to MOV-attacks, whereas anomalous curves are exposed to attacks regarding DLP.

The *bit length* of an elliptic curve $E(\mathbb{F}_q)$ refers to the bit size of q. NIST recommends a minimum of 224 bits for prime curves and 233 bits for binary curves. Prime curves are suggested to be used. The NIST prime curves are constructed over a field with specially chosen prime, allowing efficient reduction modulo p.[23] Commonly used curve NIST P-256, (or `secp256r1` in Certicom standards) is generated over a prime field with $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.[7] Certicom and NIST standards seem to agree on most of the recommendations.

Other prime curves are Montgomery and Twisted Edward curves, which are yet in limited use but garnering academic interest. Their curve type differs from the usual Weierstrass form. For instance, the Montgomery curve is defined as $Ay^2 = x^3 + Bx^2 + x,\ A, B \in \mathbb{F}_q$, where $A \neq \pm 2$ and $B \neq 0$. Montgomery and Twisted Edward curves are claimed to offer enhanced performance and resilience against side-channel attacks.[7] One alternative for standardized curves is `Curve25519` [9], which is a Montgomery curve $y^2 = x^3 + 486662x^2 + x$ over a prime field with $p = 2^{255} - 19$. It is constructed to be efficient as well as to avoid many implementation pitfalls.

Communication parties deploying ECC can agree on the used curve, and simply use the parameters in the standards. Only the random scalar value s is left as the user's responsibility (the private key). For instance, standardized curve NIST P-224 as in [7], is given by the parameters:

$p = 2^{224} - 2^{96} + 1$
$= 26959946667150639794667015087019630673557916260026308143510066298881$
($=$ `0xffffffffffffffffffffffffffffffff000000000000000000000001`)

$b = 1895828628556660800040866854449392641550468096867932107578723467256$
($=$ `0xb4050a850c04b3abf54132565044b0b7d7bfd8ba270b39432355ffb4`)

$E : y^2 \equiv x^3 - 3x + b \pmod{p}$, with an order
$\#E = 26959946667150639794667015087019625940457807714424391721682722368061$
($=$ `0xffffffffffffffffffffffffffff16a2e0b8f03e13dd29455c5c2a3d`)

which is a prime, i.e. $h = 1$. Standards also includes the base point $G = (G_x, G_y)$ and the seed value for SHA-1 algorithm, whose output is used to generate the curve parameters. Thus, the seed can be used to verify the validity of the curve parameters.[7]

In addition to the secure curve parameters, the used algorithms can fortify security. The choice of algorithms for point arithmetic is usually a trade-off between efficiency and side-channel security. Appropriate algorithms for elliptic curve arithmetic are discussed in chapter 5.

2.6.1 Algorithms

Scalar multiplication causes the most workload on elliptic curve computations. The goal is to compute kP for some $k \in \mathbb{N}$ and $P \in E$. One convenient method is the *double-and-add* approach for elliptic curve points. Let $k = k_0 + k_1 \cdot 2 + k_2 \cdot 2^2 + \ldots + k_t \cdot 2^t$, where $k_i \in \{0, 1\}$ for $i = 0, 1, \ldots, t-1$ and $k_t = 1$. The double-and-add algorithm is defined as follows:

```
   input : Elliptic curve $E$, curve point $P$ and scalar $k$ as binary representation
   output: $P' = kP$
1  Initialize $P' \leftarrow P$
2  for $i \leftarrow t-1$ to $0$ do
3      $P' \leftarrow 2P' + k_i P$
4  end
5  Return $P'$
```

Algorithm 1: Double-and-Add algorithm

The algorithm scans the bit representation of k bit by bit. Doubling is performed in every iteration, and if $k_i = 1$, addition of P is performed. On average, for a random $t+1$ bit scalar, the algorithm needs $1,5t$ point doubles and additions.[33]

Although the Double-and-Add method is efficient, it is vulnerable to side-channel attacks. As the point addition and doubling consume different amounts of power, the attacker may distinguish the value of k_j, using the Simple Power-Analysis attack.[30]

Another, more suitable algorithm for cryptographic applications is the *Montgomery ladder*, as depicted in Algorithm 2.

```
  input  : Elliptic curve point $P \in E$ and scalar $k$ as binary representation
  output: $Q = kP$
1 Initialize $R_0 \leftarrow P$, $R_1 \leftarrow 2P$
2 for $i \leftarrow t - 1$ to $0$ do
3     $b \leftarrow k_i$, $R_{1-b} \leftarrow R_{1-b} + R_b$
4     $R_b \leftarrow 2R_b$
5 end
6 Return $R_0$
```

Algorithm 2: Montgomery ladder

Montgomery ladder has several advantages for cryptographical purposes. As it performs the same curve operations in every iteration, the adversary can not inspect the bits of k from the power traces. Thus, attacks based on power-analysis are prevented. Besides, certain fault-injection attacks are thwarted, as dummy operations are not needed.[30]

Chapter 3

Pairing Based Cryptography

The field of pairing-based cryptography is the enabling factor for a new class of security schemes, such as identity-based cryptosystems [11], searchable encryption [26] and certificateless signatures [34]. Originally, pairings were invented for purposes of cryptanalysis [27], utilizing the Weil pairing. Pairings have been adopted as an indispensable tool for protocol designers, and a lot of contribution is given for enhancing their performance. Currently, one of the most efficient pairings is the optimal ate pairing by Vercauteren [38], which is used in this work.

The fundamental idea of bilinear pairings is to construct a mapping between two convenient cryptographic groups, reducing a problem in one group to a different, usually easier task in another group. The bilinear pairing itself is not directly associated with elliptic curves. However, the majority of the pairings utilize subgroups or quotient groups of elliptic curves over prime fields or prime field extensions. In this work, groups over pairing-friendly Barreto-Naehrig [8] curves are employed.

This chapter serves as an introduction to pairing based cryptography, containing the generalized notion of the bilinear pairing, some security aspects, and an example of pairing-based protocol; the Boneh-Boyen cryptosystem. Furthermore, the optimal ate pairing and the related concepts are presented. The main sources are [30], [18] and [38].

Note: due to frequent usage of ∞ as the neutral element in the relating literature, it is henceforth used as the neutral element instead of $\mathcal{O}$.

3.1 Bilinear Pairings

Let G_1, G_2 be two groups of prime order n, under a group operation $\circ$, and G_T a cyclic multiplicative group of the order n. A *bilinear pairing*, (or simply, a *pairing*) is an efficiently computable mapping $e : G_1 \times G_2 \to G_T$ satisfying the following conditions ([30]):

1. Bilinearity: Mapping e is *linear* in both arguments: $\forall P, Q \in G_1, \forall P', Q' \in G_2$:

$$e(P \circ Q, P') = e(P, P')e(Q, P')$$
$$e(P, P' \circ Q') = e(P, P')e(P, Q').$$

2. Non-degeneracy: For any $P \in G_1$ exists $Q \in G_2$ for which $e(P, Q) \neq 1$, and for any $Q \in G_2$ exists $P \in G_1$ such that $e(P, Q) \neq 1$.

The operation $\circ$ for groups G_1 and G_2 is usually addition of elliptic curve points, but it can as well be a multiplicative operation. From the bilinearity it follows that $e(aP, bQ) = e(P, Q)^{ab}$, $\forall a, b \in \mathbb{Z}$. It is possible that $G_1 = G_2$, the pairing is then symmetric. For a symmetric pairing: $e(P, Q) = e(Q, P)$.[30]

As previously mentioned, majority of pairings are constructed from elliptic curves. Considering an elliptic curve E with an order of $\#E = n \cdot h$. Now if n is a prime, then G_1 is the subgroup of $E(\mathbb{F}_q)$ of order n: $G_1 = \{hP \mid P \in E(\mathbb{F}_q)\}$. Then for a proper choice of k, there exists a pairing $e : G_1 \times G_2 \to G_T$, where $G_2 \subset E(\mathbb{F}_{q^k})$ and $G_T \subset \mathbb{F}^*_{q^k}$. It follows that $\#G_T|(q^k - 1)$.[18]

The actual computation of a certain pairing depends on the type of the pairing and the used curve type. As mentioned, the idea is to form an efficient mapping to transform a problem in a certain group into an easier problem in another group. The ate pairing and it's optimized variant is presented in sections 3.5 and 3.6.

3.1.1 Security of Bilinear Pairings

The following property is closely related to security but also in the efficiency of the computation of pairing:

Definition 3.1 ([18]). *Embedding degree. Let $P \in E(\mathbb{F}_q)$ be a point of prime order n such that* $\gcd(n, q) = 1$. *The* embedding degree *of $\langle P \rangle$ is the smallest $k \in \mathbb{Z}_{>0}$ such that $n \mid q^k - 1$.*

As mentioned, the curves with low embedding degree are exposed to the MOV attack. The MOV attack utilizes Weil-pairing to transform an instance of ECDLP into a corresponding instance of DLP over $G_T \subset \mathbb{F}_{q^k}^*$, where k is the embedding degree. Let $P, aP \in G_1$ and $Q \in G_2$. Then compute $g = e(P,Q)$ and $h = e(\alpha P, Q)$. Now by linearity: $h = e(P,Q)^\alpha = g^\alpha$. Thus, computing the discrete logarithm of αP in G_1 is equal to finding the dicrete logarithm of g^α in G_T. Hence the index calculus algorithm can be applied to attack the system.[18][27]

The intractability of Bilinear Diffie-Hellman problem is the underlying concept for secure pairing-based protocols. It is formulated as follows.

Definition 3.2 ([18])**.** *Bilinear Diffie-Hellman problem. Let $e : G_1 \times G_2 \to G_T$ be a bilinear pairing. The* bilinear Diffie-Hellman problem (BDHP) *is defined as: Given $P, aP, bP \in G_1$ and $Q \in G_2$, compute $e(P,Q)^{ab}$.*

Hardness of BDHP implies hardness of DHP in both G_1 and G_T. That is, computing of abP is difficult for known aP and bP. Moreover in G_T, finding $g^a = e(aP,Q)$, $g^b = e(bP,Q)$ and then computing g^{ab}.[18]

To choose an appropriate curve for pairing operations, the parameters q, n and k of the relation $n|q^k - 1$ should meet certain conditions. Prime divisor n of $\#E(\mathbb{F}_q)$ should be sufficiently large to preclude the attack with Pollard's rho. The embedding degree k should be large enough to thwart the MOV-attack. Cryptographically weak supersingular curves have embedding degree $k \in \{1, 2, 3, 4, 6\}$. On the other hand, k should be small enough so that pairing with $\mathbb{F}_{q^k}$ can be computed efficiently. In addition, low Hamming weight of n speeds up the doubling operations.[18][30]

Curves satisfying these properties are called *pairing friendly.* The vast majority of elliptic curves have an extremely large embedding degree, rendering the pairing computation infeasible, as well as the MOV attack.[18] Pairing friendly curves are relatively rare, as $k \approx n$ for a majority of curves as shown in the work of Balasubramanian et al. [5] for prime-order elliptic curves over prime fields.

3.2 Barreto-Naehrig Curves

Barreto and Naehrig [8] composed a method for generating pairing-friendly elliptic curves with prime order and embedding degree $k = 12$. The equation of the *Barreto-Naehrig (BN)* curve is $E : y^2 = x^3 + b,\ b \neq 0$. BN-curves utilize a certain parametrisation of the trace of Frobenius of the curve, the curve order $\#E$ and the group order

$\#\mathbb{F}_q$, as follows:

$$\begin{aligned} t(u) &= 6u^2 + 1 \\ p(u) &= 36u^4 + 36u^3 + 24u^2 + 6u + 1 \\ n(u) &= 36u^4 + 36u^3 + 18u^2 + 6u + 1. \end{aligned}$$

Due to parameterization, the memory needed for information of a BN curve is small: only the parameter u suffices. Parameter u must be chosen such that both $n = n(u)$ and $p = p(u)$ are prime. Parametrisation also enables more efficient computations.[8][30]

Another benefit from BN-curves is that they possess an efficiently computable group homomorphism, which allows computing points of $E(\mathbb{F}_{p^{12}})$ as points in $E'(\mathbb{F}_{p^2})$. The group $E'(\mathbb{F}_{p^2}) : y'^2 = x'^3 + b/\xi$ is the *sextic twist* of $E(\mathbb{F}_{p^{12}})$, where ζ is an element for which $x^6 - \zeta$ is irreducible over $\mathbb{F}_{p^2}[x]$. It is used to construct the $\mathbb{F}_{p^{12}}$ over the second-degree field $\mathbb{F}_{p^2}$. The injective group homomorphism is defined as $\psi : E'(\mathbb{F}_{p^2}) \rightarrow E(\mathbb{F}_{p^{12}})$, $(x', y') \mapsto (x'z^2, y'z^3)$. The mapping produces a point whose x-coordinate is in $\mathbb{F}_{p^6}$, and y-coordinate in $\mathbb{F}_{p^4}$.[8]

Sextic twist, or generally, a degree-d twist is a certain transformation of an elliptic curve. The process is also known as *point-compressing.* It allows pairing values to be compressed to one-third or even one-sixth of their original length.[8]

3.3 Boneh-Boyen Identity Based Encryption

In the usual public key infrastructure, generating and distributing the public keys require an agreement between communication parties. This furthermore leads to a need of certificates to verify the ownership of a certain public key.

In the Identity Based Encryption (IBE) the public key is constituted by a certain predetermined string. The corresponding private key is generated by (and only by) the Private Key Generator (PKG), which knows the certain master secret. Each user of the system has some value bound to his identity, for example, an e-mail address, from which the public key is derived. The user authenticates himself to PKG, which assigns him the private key, corresponding to his identity.[11]

The first applicable version of identity-based cryptosystem was brought up by Boneh and Franklin [12] with their solution using the Weil-pairing. Another, more secure[1] version is the Boneh-Boyen IBE algorithm [11], which is represented next.

[1]Boneh-Boyen does not require the random-oracle assumption, whereas the former Boneh-Franklin

Boneh-Boyen IBE utilizes collision-resistant hash-functions and its security is based on Decisional BDH assumption. Boneh-Boyen IBE makes use of a symmetric pairing $G_1 \times G_1 \to G_T$, where G_1 and G_T are **multiplicative** groups. Let $\sum = \{0, \dots, s\}$ be an alphabet with size of s, and $\{H_k : \{0,1\}^w \to \sum^n\}_{k \in K}$ with $w > 0$ be a family of hash functions. The identification (ID) is assumed to be a w-length bit string $\{0,1\}^w$. The algorithm has four different operations: setup, key generation, encryption, and decryption.

Setup. First, a random generator $g \in G_1^*$ and $\alpha \in \mathbb{Z}_p$ are chosen and $g_1 = g^\alpha$ is set. Next, a random $g_2 \in G_1$ is chosen and a random $n \times s$ matrix $U = (u_{i,j}) \in G^{n \times s}$ is constructed, where each $u_{i,j}$ is uniform in G. Moreover, a random $k \in K$ is picked as the key for the hash function. Now the system parameters are (g, g_1, g_2, U, k), and the master-key is g_2^α.

Key generation. Denote $\mathbf{a} = H_k(ID) = (a_1 \dots a_n) \in \sum^n$ (n-length vector of alphabets) and pick random $r_1, \dots, r_n \in \mathbb{Z}_p$. The private key corresponding an identity $ID \in \{0,1\}^w$ is generated as

$$d_{ID} = \left(g_2^\alpha \cdot \prod_{i=1}^{n} u_{i,a_i}^{r_i},\ g^{r_1},\ \dots, g^{r_n} \right) \in G^{n+1}.$$

Encryption. Let $M \in G_1$ be a message and $\mathbf{a}$, and $H_k(ID)$ as above and let t be a random element in $\mathbb{Z}_p$. Now the encryption of M under the given parameters and the public ID is as follows:

$$C = \left(M \cdot e(g_1, g_2)^t,\ g^t,\ , u_{1,a_1}^t,\ \dots,\ u_{n,a_n}^t \right) \in G_T \times G_1^{n+1}.$$

The cryptotext C is thus $(n+2)$-tuple of elements in G_T and G_1. For simplification, we denote $A = M \cdot e(g_1, g_2)^t$ and $u_{j,a_j}^t = C_j$.

Decryption. The owner of the private key $d_{ID} = (d_0, d_1, \dots, d_n)$ can decrypt the message $C = (A,\ g^t,\ C_1, \dots, C_n)$ as

$$A \cdot \frac{\prod_{j=1}^{n} e(C_j, d_j)}{e(g^t, d_0)} = M.$$

Let's go through the calculations more in detail to justify the process.

The goal is to solve the M from $M \cdot e(g_1, g_2)^t$ using the information of the private key d_{ID} and elements g^t and C_j of the vector C. Now recalling the bilinearity of a pairing,

version does.

i.e. $e(P, Q \cdot R) = e(P, Q) \cdot e(P, R)$ and the notations for C_j and d_j above:

$$\begin{aligned} A \cdot \frac{\prod_{j=1}^n e(C_j, d_j)}{e(g^t, d_0)} &= A \cdot \frac{\prod_{j=1}^n e(u_{j,a_j}^t, g^{r_j})}{e(g^t, g_2^\alpha \cdot \prod_{j=1}^n u_{j,a_j}^{r_j})} = A \cdot \frac{\prod_{j=1}^n e(u_{j,a_j}, g)^{tr_j}}{e(g^t, g_2^\alpha) e(g^t, \prod_{j=1}^n u_{j,a_j}^{r_j})} \\ &= A \cdot \frac{\prod_{j=1}^n e(u_{j,a_j}, g)^{tr_j}}{e(g, g_2)^{\alpha t} e(g^t, \prod_{j=1}^n u_{j,a_j}^{r_j})} \overset{\star}{=} A \cdot \frac{\prod_{j=1}^n e(u_{j,a_j}, g)^{tr_j}}{e(g^\alpha, g_2)^t \prod_{j=1}^n e(g, u_{j,a_j})^{tr_j}} \\ &= M \cdot e(g_1, g_2)^t \frac{1}{e(g_1, g_2)^t} = M. \end{aligned}$$

The equality at ($\star$) follows from applying the bilinearity property consecutively on the pairing in the denominator. Also, the property $e(P^\alpha, Q^\beta) = e(P, Q)^{\alpha\beta} = e(P^\beta, Q^\alpha)$, and the commutativity of symmetric pairing was used, i.e. $e(P, Q) = e(Q, P)$.

For proofs of security and further details, one may refer to [11].

3.4 Divisors and Rational Functions

The notion of a *divisor* of an elliptic curve and rational functions on elliptic curves are closely related on evaluation of a pairing. All known pairings include an evaluation of a certain rational function, with a certain divisor.[18] Divisors contain information about the function of interest, namely of the special points; *poles* and *zeros*. A function is said to have a pole at P if it gets the value ∞ at that point, and a zero if it gets the value 0.

Definition 3.3 ([18]). *Divisor. A divisor on an elliptic curve $E(K)$ is a formal sum $D = \sum_{P \in E} a_P P$ of points P on the curve such that*

1. *The sum is finite,*
2. *The coefficient a_P is an integer for each point P,*
3. *Two sums equal only if the all the coefficients equal. That is, the sum is unique.*

The *degree* D of a divisor is defined as $\deg(D) = \sum_{P \in E} a_P$. The empty divisor is denoted as $\emptyset$, and it's degree is 0.[18]

Definition 3.4 ([18]). *Rational function over E. Let E be an elliptic curve (eq. 2.1) over a field K. A rational function on E is a function $f : E \to K$ of the form*

$$f(x, y) = \frac{f_1(x, y)}{f_2(x, y)}$$

where f_1 and f_2 are polynomials in the variables x and y.

Definition 3.5 ([18]). *Let $f(x,y) \neq 0$ be rational function over E as defined above. The order of f for any point $P \in E$, denoted as $ord_P(f)$ is defined as follows:*

1. *If $f(P) \neq 0$ and $\frac{1}{f(P)} \neq 0$, then $ord_P(f) = 0$.*
2. *If $f(P) = 0$, then $ord_P(f)$ is the multiplicity of the root at P of the numerator $f_1(x,y)$.*
3. *If $\frac{1}{f(P)} = 0$, then $ord_P(f)$ equals the negative of the multiplicity of the root at P of the denominator $f_2(x,y)$.*

Summarizing, the function f has non-zero order on a point P, only if the point is pole or zero of the function.

Definition 3.6 ([18]). *Principal divisor. Let $f \neq 0$ be a rational function on an elliptic curve E. The principal divisor $div(f)$, generated by f is denoted as*

$$div(f) = \sum_{P \in E} ord_P(f) \cdot (P).$$

Principal divisor expresses the finite sum of all the points $P \in E$ on which either f_1 or f_2 equal to zero. In other words, it is a linear combination of poles and zeros of the function f on elliptic curve E.

Furthermore, a divisor D on E is called a principal divisor if $div(f) = D$ for some rational function f on E. Also, the mapping $div(f)$ is a homomorphism from multiplicative to additive group: $div(fg) = div(f) + div(g)$ and $div(1) = \emptyset$.[18]

Theorem 3.1 ([18]). *For any rational function f on elliptic curve E: $\deg(div(f)) = 0$.*

Example 5. *Let $E : y^2 = x^3 - x$ be an elliptic curve, and let $f(x,y) = \frac{x}{y}$ on E. The principal divisor $div(f)$ can be computed as follows. First considering $f_1(x,y) = x$, which equals zero when $P = (0,0)$. Multiplicity of the root is 2, so $ord_{(0,0)}(f_1) = 2$. Moreover, $1/f_1 = 0$ when $P = \infty$, and by thm. (3.1): $ord_\infty(f_1) = -2$.*

Similarly for the function $f_2(x,y) = y$. As $y^2 = x^3 - x = x(x-1)(x+1)$, f_2 equals zero on $P = (0, \pm 1)$ and $P = (0,0)$. For $1/f_2 = 0$, $P = \infty$, with order of -3 (thm (3.1)).

Now by utilizing $div(f/g) = div(f) - div(g)$, the principal divisor is

$$\begin{aligned} div(f) &= \sum_{P \in E} ord_P(f) \cdot (P) = \sum_{P \in E} ord_P(x) \cdot (P) - \sum_{P \in E} ord_P(y) \cdot (P) \\ &= \{2((0,0)) - 2(\infty)\} - \{((0,0)) + ((1,0)) + ((-1,0)) - 3(\infty)\} \\ &= ((0,0)) - ((1,0)) - ((-1,0)) + (\infty) \end{aligned}$$

Thus, the function has zeros at $(0,0)$ *and* ∞*, and poles at* $(1,0)$ *and* $(-1,0)$*. Note that the function* $\frac{x}{y}$ *as itself is not defined at the point* $(x,y) = (0,0)$*. But on the elliptic curve E:* $y^2 = x^3 - x \iff \dfrac{x}{y} = \dfrac{y}{x^2-1}$*, which is defined at the point* $(0,0)$*.*

3.5 The Ate Pairing

As defined in [38], the ate pairing is a non-degenerated bilinear asymmetric pairing $G_1 \times G_2 \to G_T$, defined over the groups

$$\begin{aligned}
G_1 &= E(\mathbb{F}_p)[n] \cap Ker(\pi_p - [1]), \\
G_2 &= E(\mathbb{F}_{p^k})[n] \cap Ker(\pi_p - [p]), \\
G_T &= \mathbb{F}^*_{p^k}/(\mathbb{F}^*_{p^k})^n,
\end{aligned}$$

where $E(\mathbb{F}_{p^k})[n]$ denotes the n torsion group of $E(\mathbb{F}_{p^k})$, π_p is the Frobenius endomorphism $E \to E : (x,y) \mapsto (x^p, y^p)$ and $[n]$ denotes the scalar multiplication of a curve point.[38] The group can be interpret as follows: G_1 is r-torsion points of $E(\mathbb{F}_p)$, for which $\pi_p(P) - [1]P = 0$, i.e. $\pi_p(P) = P$. This applies for points with coordinates in non-extension field $\mathbb{F}_p$. The group G_T is a quotient group, where $(\mathbb{F}^*_{p^k})^n$ is the elements of $\mathbb{F}^*_{p^k}$ raised to the nth power.

The ate pairing for $P \in G_1$ and $Q \in G_2$ (the arguments are swapped) is defined as:

$$e(Q,P) = f_{\lambda,Q}(P)^{(p^k-1)/n}. \tag{3.1}$$

For derivation of the pairing and proofs for bilinearity and redundancy, see [38]. The computation of ate pairing consists of two phases: evaluation of a certain Miller function $f_{\lambda,P}(\cdot)$, and a final exponentiation process.

3.5.1 Miller's Algorithm

The *Miller function* $f_{r,P}(\cdot)$ is a rational function with r zeros at P, a pole at $[r]P$ and $r-1$ poles at ∞:

$$div(f_{r,P}) = r(P) - ([r]P) - (r-1)\infty. \tag{3.2}$$

The idea of Miller's function is to map two elliptic curve points into a single point in finite field. In general, the function can not be evaluated directly, but the following observation enables efficient computation of it.

Lemma 3.2 ([28])**.** *Let $P \in E[n]$ and $i, j \in \mathbb{N} \setminus \{0\}$. Let $l_{A,B}$ be a line through points $A = (x_A, y_A)$ and $B = (x_B, y_B)$, and v be a vertical line through a point $A + B$, where $A = [i]P$ and $B = [j]P$. Then*

$$f_{i+j,P} = f_{i,P} f_{j,P} \frac{l_{[i]P,[j]P}}{v_{[i+j]P}}. \tag{3.3}$$

Proof. Using the fact of eq. (3.2) and that the divisor is a homomorphism:

$$\begin{aligned}
div(f_{i,P} f_{j,P} \frac{l}{v}) &= div(f_{i,P}) + div(f_{j,P}) + div(l) - div(v) \\
&= \{i(P) - ([i]P) - (i-1)(\infty)\} + \{j(P) - ([j]P) - (j-1)(\infty)\} \\
&+ \{([i]P) + ([j]P) + (-[i+j]P) - 3(\infty))\} \\
&- \{([i+j]P) + (-[i+j]P) - (i+j-1)(\infty)\} \\
&= (i+j)(P) - ([i+j]P) - (i+j-1)(\infty) \\
&= div(f_{i+j})
\end{aligned}$$

□

As a consequence, the function $f_{r,P}(\cdot)$ can be constructed efficiently with Miller's algorithm, as depicted in Algorithm 3, from [38]. The algorithm computes the function with $\mathcal{O}(\log n)$ steps, utilizing the double-and-add approach.[28] For further details and more rigorous reasoning of the presented information, one can refer to the original article by Miller [28].

```
   input : $P, Q \in E[n]$, $P \neq Q$ and $r \in \mathbb{N}$.
   output: $f_{r,P}(Q)$.
1  Let $r = \sum_{i=0}^{L} r_i 2^i$, with $r_i \in \{0,1\}$ and $r_L = 1$.
2  $T \leftarrow P$, $f \leftarrow 1$
3  for $i \leftarrow L-1$ to $0$ do
4      $f \leftarrow f^2 \cdot \frac{l_{T,T}(Q)}{v_{[2]T}(Q)}$
5      $T \leftarrow [2]T$
6      if $r_i = 1$ then
7          $f \leftarrow f \cdot \frac{l_{T,P}(Q)}{v_{T \oplus P}(Q)}$
8          $T \leftarrow T \oplus P$
9      end
10 end
11 return $f$
```

Algorithm 3: Miller's algorithm for computing $f_{r,P}(Q)$

One common optimization in Miller function computation is the *denominator elimination*. With certain conditions, the element $v_{T \oplus P}(Q)$ is in $\mathbb{F}_{q^d} \subset \mathbb{F}_{q^k}$. As the order of the subfield divides the order of the field $\mathbb{F}_{q^k}$, the denominator is eliminated in final exponentiation. Therefore, computation of it can be ignored. Namely if $x \in \mathbb{F}_{q^d}$, then $x^{(q^k-1)/n} = 1$. This applies especially when embedding degree k is even.[30]

3.5.2 Final Exponentiation

The final exponentiation process ensures the unique result of the pairing. The output of the Miller loop is raised to the power of $(p^k - 1)/n$ to obtain an unique representative of an element in the coset $\mathbb{F}^*_{p^k}/(\mathbb{F}^*_{p^k})^n$. Let $k = 2d$, then:

$$\frac{p^k - 1}{n} = (p^d - 1) \cdot \frac{p^d + 1}{\Phi_k(p)} \cdot \frac{\Phi_k(p)}{n},$$

where $\Phi_k(p)$ is a certain *k:th cyclotomic polynomial*. The first two coefficients of the exponentiation are called the easy part of the final exponentiation, whereas the coefficient $\Phi_k(p)/n$ forms the hard part.[30]

In the context of BN curves, the exponential $(p^{12} - 1)/n$ is factorized into $(p^6 - 1)$, $(p^2 + 1)$ and $(p^4 - p^2 + 1)/n$. The first two terms form the easy part, which is easily computed owing to cheap Frobenius mapping. The hard part, i.e. the exponentiation by $(p^4 - p^2 + 1)/n$ needs extra treatment, as the fraction will be decomposed into a large polynomial by expanding the parametrizations.

The process of calculating depends of the used curve and fields. In this work, the modified variant of final exponentiation by Fuentes-Castaneda [20] for BN curves are used, in addition to squaring in cyclotomic subgroups [21]. The final exponentiation process is inspected more in detail in the section 4.4.

3.6 Optimal Ate pairing over Barreto-Naehrig Curves

Vercauteren [38] presented an idea of constructing optimal pairings, with a reduced Miller loop length $(1/\varphi(k)) \log_2 n$, where k is the embedding degree. The idea of optimal ate pairing (or O-ate) is on finding base-q expansion for $\lambda = \sum_{i=0}^{l} c_i q^i$, divisible by n. The optimal bilinear pairing is then given as

$$(P,Q) \mapsto \left(\prod_{i=0}^{l} f_{c_i,Q}^{q^i}(P) \cdot \prod_{i=0}^{l-1} \frac{l_{[s_{i+1}]Q,[c_i q^i]Q}(P)}{v_{[s_i]Q}(P)} \right)^{(q^k-1)/n} \tag{3.4}$$

where $s_i = \sum_{j=i}^{l} c_j q^j$. It is required that $mkq^{k-1} \not\equiv ((q^k-1)/n) \cdot \sum_{i=0}^{l} i c_i q^{i-1} \pmod{n}$ for the pairing to be non-degenerate.

Vercauteren showed that by choosing a polynomial λ with small coefficients c_i, the Miller loop has at most $(1/\varphi(k)) \log_2 n$ iterations. Barreto-Naehrig curves can be utilized in this purpose.

Recalling the curve order and the group order for BN curves: $p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$ and $n(u) = 36u^4 + 36u^3 + 18u^2 + 6u + 1$. By direct computation it can be observed that

$$(6u+2) + p(u) - p(u)^2 + p(u)^3 \equiv 0 \pmod{n(u)}.$$

This represents the $q = p(u)$ based expansion for λ, as it also divides the $n(u) = \#\mathbb{F}_{p^k}$. As the BN curves are used, the denominator from $l_{[s_{i+1}]Q,[c_i q^i]Q}(P)/v_{[s_i]Q}(P)$ can be eliminated. Now by applying λ in 3.4, an optimal ate pairing over BN curve is obtained:

$$\begin{aligned}(Q,P) \mapsto &(f_{6x+2,Q}(P) f_{1,Q}^{p}(P) f_{-1,Q}^{p^2}(P) f_{1,Q}^{p^3}(P) \\ &\cdot l_{[p-p^2+p^3]Q,[6x+2]Q}(P) l_{[-p^2+p^3]Q,[p]Q}(P) l_{[p^3]Q,[-p^2]Q}(P))^{(p^k-1)/r} \\ &= \big(f_{6x+2,Q}(P) \cdot l_{[p-p^2+p^3]Q,[6x+2]Q}(P) l_{[-p^2+p^3]Q,[p]Q}(P) l_{[p^3]Q,[-p^2]Q}(P)\big)^{(p^k-1)/r}\end{aligned}$$

The equality follows from $f_{1,Q}(P) = f_{-1,Q}(P) = 1$. Now with a further fine-tuning, as in [31], the final version of optimal ate pairing is

$$a_{opt} = (f_{6x+2,Q}(P) \cdot l_{[6x+2]Q,\pi_p(Q)}(P) \cdot l_{[6x+2]Q+\pi_p(Q),-\pi_p^2(Q)}(P))^{(p^{12}-1)/r} \tag{3.5}$$

where $\pi_p(Q)$ is the Frobenius endomorphism.

Chapter 4

High Level Arithmetic

As mentioned, the pairing implementation contains many sub-processes that can be optimized further to obtain a more enhanced version of the pairing. As demonstrated in Figure 4.1, the arithmetic of bilinear pairings occurs in several layers. Miller's loop phase constructs the rational Miller function using arithmetic of elliptic curve points. The curve operations, as well as the final-exponentiation process, ultimately break down into operations in large extension fields. These consecutively reduce into prime-field arithmetic, and eventually into machine instructions with large integers.

The used pairing implementation by Unterluggauer *et al.* [37] follows the state-of-art design of [10] and [16]. The efficient formulas by Costello *et al.* [15] are used for curve point arithmetic in the Miller loop. For final exponentiation, a modified variant by Fuentes-Castaneda *et al.* [20], and efficient squaring formulas in cyclotomic subgroups [21] are used. Also, in the O-ate pairing function itself, an inversion trick by Aranha *et al.* [1] is utilized.

This chapter presents the key methodology employed within the O-ate pairing over the BN curves, keeping the scope on the methods relevant for this work. In section 4.1, the arithmetic on the finite fields, the construction of extension fields, and some optimizing tricks are outlined. The second section presents the used curve point arithmetic. Section 4.3 explains the computations used in the final exponentiation, containing less memory consuming optimization.

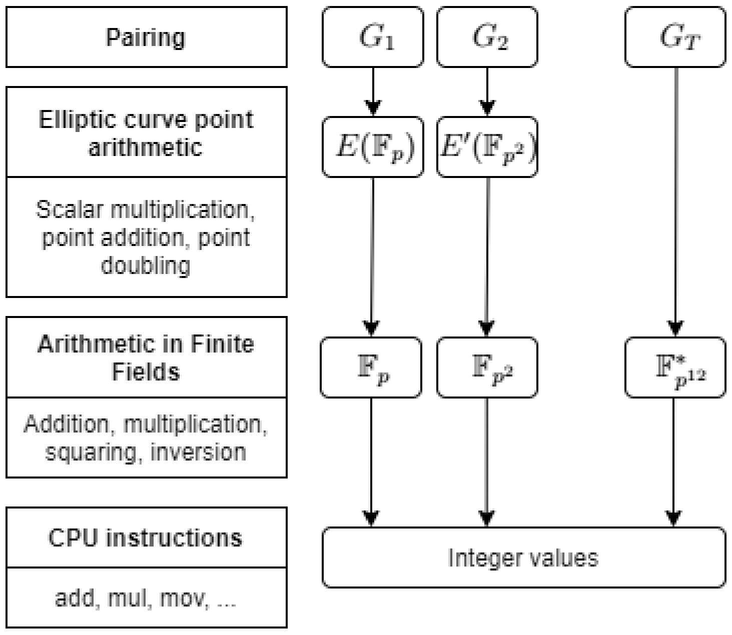

Figure 4.1: Operation flow of the pairing process

4.1 Field Arithmetic

Careful implementation of field and integer arithmetic has a crucial role in achieving efficient pairing implementations. The operations with field extensions such as $\mathbb{F}_{p^{12}}$ and $\mathbb{F}_{p^2}$ are reduced into operations on the prime field $\mathbb{F}_p$. These, in turn are processed as a set of word-size multi-precision integer values and computed as CPU instructions.

In this section, we outline the basic ideas of field arithmetic and present the essential methods used in this work. The implementation aspects in hardware level are also considered. The field multiplication is emphasized, as it will be optimized in this work at the instruction level. The field arithmetic is mainly based on the techniques used by Beuchat *et al.* [10] , Devegili *et al.* [16] and Aranha *et al.* [1]. Another supportive literature used is [30].

4.1.1 Extensions Fields

To perform extension field arithmetic efficiently, the high degree field extensions are constructed as a *tower* of lower degree sub-fields. The main idea is expressing the prime field extension $\mathbb{F}_{p^k} = \mathbb{F}_p[z]/f(z)$ as $\mathbb{F}_{p^m}$, where $m|k$ such that

$$\begin{aligned}\mathbb{F}_{p^k} &= \mathbb{F}_{p^m}[v]/g(v),\ where\ g(v) \in \mathbb{F}_{p^m}[v]\ ,\ \deg(g) = k/m \\ \mathbb{F}_{p^m} &= \mathbb{F}_p[u]/h(u),\ where\ h(u) \in \mathbb{F}_p\ ,\ \deg(h) = m,\end{aligned}$$

also recalling that $g(v)$ and $h(u)$ must be irreducible polynomials. Low-degree polynomials of low hamming weight are commonly used as the irreducible polynomials, as they provide efficient multiplications and reductions.[30]

The used BN-curve parameter in this work is $u = -2^{62} - 2^{55} - 1$, which applied to the parametrization, yields $p(u) \equiv 3 \pmod 4$. Therefore $x^2 - \beta$, where $\beta = -1$ can be used as the irreducible polynomial for $\mathbb{F}_{p^2}$. This results in faster arithmetic, as the multiplication by β corresponds to a simple subtraction.[1]

In order to utilize the sextic twist of BN curves, the field $\mathbb{F}_{p^{12}}$ must be built as an extension of $\mathbb{F}_{p^2}$ for certain operations. The following tower extensions are used in this work:

$$\begin{aligned}\mathbb{F}_{p^2} &= \mathbb{F}_p[u]/(u^2+1) \\ \mathbb{F}_{p^4} &= \mathbb{F}_{p^2}[s]/(s^2-\zeta),\ where\ \zeta = 1+i \\ \mathbb{F}_{p^{12}} &= \mathbb{F}_{p^2}[v]/(v^6-\zeta)\ or\ \mathbb{F}_{p^{12}} = \mathbb{F}_{p^4}[v]/(v^3-s)\end{aligned}$$

The towering $\mathbb{F}_p \to \mathbb{F}_{p^2} \to \mathbb{F}_{p^4} \to \mathbb{F}_{p^{12}}$ is mostly adopted for arithmetic in this work, but another variant can be applied by permuting the order of coefficients.[1][37]

In the hardware level, a field element $x \in \mathbb{F}_p$ is presented according to the size of p and the word length W of the processor in bits. In this work p is a size of 256 bits and $W = 32$. An arbitrary $x \in \mathbb{F}_p$ is presented in the memory as $x_7 2^{32\cdot 7} + x_6 2^{32\cdot 6} + x_5 2^{32\cdot 5} + x_4 2^{32\cdot 4} + x_3 2^{32\cdot 3} + x_2 2^{32\cdot 2} + x_1 2^{32} + x_0$, or $(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0)$, where x_i is a size of under 32 bits. An element x of an extension field $\mathbb{F}_{p^k}$ would consist of k aforementioned 8-word arrays of prime field elements. For more details, one might refer to [30].

4.1.2 Prime Field Multiplication

An efficient prime field multiplication is a standard building block of the pairing function. The general method for multiplying $a, b \in \mathbb{F}_p$ consists of computing $t = a \cdot b$ and then applying reduction, i.e. computing $t \pmod p$. In this work, efficient Karatsuba multiplier paired with Montgomery multiplication is used.

The Karatsuba multiplier exploits a certain base presentation and "divide and conquer" approach, allowing the number of required word-multiplications to be reduced. Let $a = (a_1, a_0)$ and $b = (b_1, b_0)$ be two integers with a bit size of $2W$. The Karatsuba product can be written as:

$$\begin{aligned} a \cdot b &= (a_1 2^W + a_0) \cdot (b_1 2^W + b_0) \\ &= a_1 b_1 2^{2W} + (a_1 b_0 + a_0 b_1) 2^W + a_0 b_0 \\ &= a_1 b_1 2^{2W} + ((a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1) 2^W + a_0 b_0 \end{aligned} \tag{4.1}$$

As can be observed, the last expression requires only three word multiplications: $a_0 b_0$, $a_1 b_1$, $(a_0 + a_1)(b_0 + b_1)$, which is less than in the basic "schoolbook" multiplication. For n-word multiplications, the complexity of Karatsuba is $\mathcal{O}(n^{\log_2 3})$, whereas the "Schoolbook" method requires $\mathcal{O}(n^2)$ steps. Karatsuba can be effectively used recursively for larger inputs.[30]

To scale the result to $\mathbb{F}_p$, a reduction is needed. This could be achieved with a costly division by p and taking the remainder. Fortunately, more efficient methods exist.

The Montgomery multiplication takes advantage of a certain mapping, which allows trading the division by p with a division by $R = 2^k$. The *Montgomery representation* is the mapping $\tilde{a} = a \cdot R \pmod{p}$, which maps the element a into a *Montgomery domain*. The Montgomery product for $c = a \cdot b \pmod{p}$ is defined as $\tilde{c} = \tilde{a} \cdot \tilde{b} \cdot r^{-1} \pmod{p}$. The result is easily mapped back to the integer domain as $c = \tilde{c} \cdot r^{-1} \pmod{p}$. The products can be computed with e.g. Karatsuba multiplier.[30]

The algorithm 4 demonstrates the classical variant of Montgomery multiplication, which additionally requires a parameter p' such that $R \cdot R^{-1} - p \cdot p' = 1$, i.e $p'p \equiv -1 \pmod{p}$. First (step 1), the elements are mapped into the Montgomery domain. Then, a certain parameter q is computed (step 2), followed by the reduction (step 3). The value is then scaled, and returned.[30] The variable $q = t \cdot p' \pmod{R}$ ensures that $t + q \cdot p$ is a multiple of R, as $t + q \cdot p \equiv t + tp' \cdot p \equiv t - t \equiv 0 \pmod{R}$. Now, as $u \cdot R \equiv t + q \cdot p \equiv t \pmod{p}$, it follows that $u = t \cdot R^{-1} \equiv \tilde{a} \cdot \tilde{b} \cdot R^{-1} \pmod{p}$ as required. After finishing, the value u must be mapped back to the integer domain.[30]

```
input  : $n$-word prime $p$, $R = 2^{n \cdot W}$, parameters $\tilde{a}, \tilde{b}, p'$
output: $u = MontPr(\tilde{a}, \tilde{b}) = \tilde{a} \cdot \tilde{b} \cdot R^{-1} \pmod{p}$.
1 $t \leftarrow \tilde{a} \cdot \tilde{b}$
2 $q \leftarrow t \cdot p' \pmod{R}$
3 $u \leftarrow (t + q \cdot p)/R$
4 if $u > p$ then
5     return $u - p$
6 else
7     return $u$
8 end
9 return $u$
```

Algorithm 4: Montgomery multiplication

The division by 2^k is a cheap operation, as it corresponds to a k-bit shift in the CPU level. The variable $k \in \mathbb{Z}$ such that $2^{k-1} < |p| < 2^k$ is required to be precomputed, as well as the parameter p'. The Montgomery multiplication has little bit of overhead due to conversion between Montgomery and integer domain, and from the computation of q. However, it is not significant as the required multiplications and additions can be performed sequentially without the need to move back to the integer domain. The reduction is especially useful in exponentiation with the square-and-multiply approach.[30]

4.1.3 Prime-Field Inversion

This work utilizes an optimized prime-field inversion. The parametrization of the BN-curve enables the usage of an exponent-based inversion instead of the common Euclidean algorithm. The algorithm makes use of Fermat's little theorem, and the parametrization of p. Applying these, the inverse of $a^{-1} \in \mathbb{F}_p$ can be expressed as

$$\begin{aligned} a^{-1} \pmod{p} = a^{p(u)-2} \pmod{p} &= a^{36u^4+36u^3+24u^2+6u-1} \pmod{p} \\ &= a^{6u-1} \cdot a^{24u^2} \cdot a^{36u^3} \cdot a^{36u^4} \pmod{p} \end{aligned}$$

As the exponents are public information, the side-channel protected, and thus more costly exponential algorithms are not needed. The exponentiation is carried out efficiently by a certain chain of computations. A clear benefit is gained from the low hamming weight of the parameter u. Additionally, the parameter $6u - 1$ can be precomputed.[37]

4.1.4 Extension Field Arithmetic

As mentioned, the computations of high-degree fields are reduced into lower degree arithmetic, ultimately into big-integer operations. The main optimization from this standpoint is to optimize the high-degree arithmetic to consume the minimum amount of such primitive operations. It is best achieved with the combination of the curve parameters the pairing function is constructed with, and the efficient field arithmetic.

As already noted, the parameter $u = -2^{62} - 2^{55} - 1 = -\texttt{40800000 00000001}_{\texttt{h}}$, in addition to the sextic twist of the BN-curve family provide optimizations. The low-hamming weight parameter provides efficient computations involving $p(u)$, and efficient tower construction. The sextic-twist, in turn, provides savings due to point-compression, recalling from section 3.2. This allows to use "sparse" multiplication as the point in $E(\mathbb{F}_{p^{12}})$ can be presented as smaller elements; in this work as elements of $\mathbb{F}_{p^2}$ and $\mathbb{F}_{p^4}$.

The extension field arithmetic follows the design philosophy of Beuchat *et al.* [10] and Devegili *et al.* [16]. The multiplication in $\mathbb{F}_{p^2}$ utilizes a lazy reduction as in [1] for multiplying. It computes the arithmetic at the big-integer level, and performs the reductions directly in the $\mathbb{F}_{p^2}$. Computationally it corresponds to 3 integer multiplications, 5 additions, and 2 reductions, which is less than in a plain Karatsuba implementation. The inversion in $\mathbb{F}_{p^2}$ is computed as $(a+bu)^{-1} = (a-bu)\cdot(a^2-\beta b^2)^{-1} = (a-bu)\cdot(a^2+b^2)^{-1}$. In $\mathbb{F}_{p^{12}}$, the Chung-Hasan complex squaring is utilized. Karatsuba multiplier is extensively used in all fields, except for the sparse multiplication in $\mathbb{F}_{p^{12}}$. In the final exponentiation process the fast squaring in cyclotomic subgroups is used. This will be briefly discussed in section 4.4.1.

Table 4.1 summarizes the costs of the basic operations over fields used in this work, in addition to the cyclotomic subgroup $\mathbb{G}_{\Phi_6} < \mathbb{F}_{p^{12}}$. The sets (a, m, s, i) and $(\hat{a}, \hat{m}, \hat{s}, \hat{i})$ correspond to addition, multiplication, squaring and inversion in the fields $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ respectively. Notable is that $\hat{m}_\zeta$ corresponds to only 1 negations and 2 additions, i.e. 3 additions computationally.

Table 4.1: Computational costs of the field extension arithmetic

Field	Add./Sub.	Multiplication	Squaring	Inversion
$\mathbb{F}_{p^2}$	$\hat{a} = 2a$	$\hat{m}$	$\hat{s} = 2m + 3a$	$\hat{i} = 2m + 2s + 3a + i$
$\mathbb{F}_{p^4}$	$2\hat{a}$	$3\hat{m} + \hat{m}_\zeta + 5\hat{a}$	$2\hat{m} + 2\hat{m}_\zeta + 5\hat{a}$	$2\hat{m} + 2\hat{s} + \hat{m}_\zeta + 2\hat{a} + \hat{i}$
$\mathbb{F}_{p^{12}}$	$6\hat{a}$	$18\hat{m}+8\hat{m}_\zeta+60\hat{a}$	$11\hat{m}+11\hat{m}_\zeta+45\hat{a}$	$35\hat{m} + 2\hat{s} + 22\hat{m}_\zeta + 70\hat{a} + \hat{i}$
$\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$	$6\hat{a}$	$18\hat{m}+8\hat{m}_\zeta+60\hat{a}$	$6\hat{m} + 7\hat{m}_\zeta + 42\hat{a}$	Conjugation

4.1.5 Frobenius Operator

In this section, the concept of *Frobenius operator* is briefly outlined. Frobenius operator is the Frobenius mapping with prime-field characteristic p; $\pi(x) = x^p$ in an extension field $\mathbb{F}_{p^d}$. It is especially used in the final exponentiation phase.

Considering the field $\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2+1)$. As $u^2 + 1 = 0$, it follows that $u = i = \sqrt{-1}$. Thus, an element $x \in \mathbb{F}_{p^2}$ can be presented as $x = a + ib$. Now observing the exponential:

$$\begin{aligned} x^p &= (a+ib)^p = a^p + i^p b^p = a + i^p b \\ &= a + i^3 b && (\text{As } p \equiv 3 \pmod 4) \\ &= a - ib = \bar{x}. \end{aligned}$$

Hence, the Frobenius mapping in the given extension field corresponds to one conjugation, which is computationally equivalent to a prime field addition. Furthermore, it is easy to verify that $x^{p^{2n}} = x$ and $x^{p^{2n-1}} = \bar{x}$ for every positive integer n and $x \in \mathbb{F}_{p^2}$.

Now using the extension $f \in \mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[v]/(v^6 - \zeta)$, also noting that $p \equiv 1 \pmod 6$, one can write $(v^p)^j = (v^{6\cdot(p-1)/6})^j v^j = \zeta^{j\cdot(p-1)/6} v^j = \zeta_j v^i$. Thus f^p can be computed as

$$\begin{aligned} f^p =& (a_0 + a_1 v + a_2 v^2 + a_3 v^3 + a_4 v^4 + a_5 v^5)^p \\ & \bar{a}_0 + \bar{a}_1 v^p + \bar{a}_2 v^{2p} + \bar{a}_3 v^{3p} + \bar{a}_4 v^{4p} + \bar{a}_5 v^{5p} \\ & \bar{a}_0 + \bar{a}_1 \zeta_1 v + \bar{a}_2 \zeta_2 v^2 + \bar{a}_3 \zeta_3 v^3 + \bar{a}_4 \zeta_4 v^4 + \bar{a}_5 \zeta_5 v^5. \end{aligned}$$

The expression above costs 6 prime-field additions and 5 multiplications in $\mathbb{F}_{p^2}$. A similar approach works for powers such as f^{p^2} and f^{p^3}, which are required in the hard part of the final exponentiation.[10]

4.2 Curve Arithmetic

The Miller's algorithm consists of curve point doubling and additions, line evaluations, and squaring and multiplications in the extension field $\mathbb{F}_{p^{12}}$. The algorithm 5 below describes the Miller's algorithm in this work. The curve arithmetic in the Miller loop is computed with efficient formulas by Costello et al. [15]. The formulas take advantage of the curve twist of the BN curve and projective coordinates.

In order to speed up computations, the line functions are computed simultaneously with the curve point doubling and addition.

The homogeneous projective coordinates (X, Y, Z), $Z \neq 0$ represent the affine point $(X/Z, Y/Z)$ with $x \mapsto X/Z$ and $y \mapsto Y/Z$. Similarly, the affine point (x, y) accepts the homogeneous coordinates $(X, Y, 1)$. The use of the homogeneous coordinates removes the need for costly division, i.e. inversions in finite fields on calculating the slope λ, thus resulting in faster elliptic curve point addition and doubling. For further justification, see [15].

Using the homogeneous coordinates, the curve equation $y^2 = x^3 + b$ becomes $Y^2Z = X^3 + bZ^3$. For computing the point doubling, line addition and point addition, the efficient formulas by Costello et al. [15] are used. The formulas utilize projective coordinates and take advantage of the sextic twist of BN-curves. Let $P = (X_1, Y_1, Z_1) \in E'(\mathbb{F}_{p^2})$. Then $2P = (X_3, Y_3, Z_3)$ is given as:

$$\begin{cases} X_3 = 2X_1Y_1(Y_1^2 - 9bZ_1^2) \\ Y_3 = Y_1^4 + 18bY_1^2Z_1^2 - 27b^2Z_1^4 \\ Z_3 = 8Y_1^3Z_1 \end{cases}$$

The corresponding line function, evaluated at $S = (x_s, y_s)$ is as follows:

$$l_{2P}(S) = 3X_1^2 \cdot x_s - 2Y_1Z_1 \cdot y_S + 3bZ_1^2 - Y_1^2$$

The point (X_3, Y_3, Z_3) and the line function, hereafter denoted as $l_{2P}(S) = L_1 \cdot x_s - L_2 \cdot y_s + L_3$, are computed with the following steps:

$$\begin{aligned} &A = X_1^2, \quad B = Y_1^2, \quad C = Z_1^2, \quad D = 3bC, \quad E = (X_1 + Y_1)^2 - A - B, \\ &F = (Y_1 + Z_1)^2 - B - C, \quad G = 3D, \quad X_3 = E(B - G), \quad Y_3 = (B + G)^2 - 12D^2, \\ &Z_3 = 4BF, \quad L_1 = 3A, \quad L_2 = -F, \quad L_3 = D - B. \end{aligned}$$

Addition is performed with mixed use of affine and projective coordinates to obtain the best efficiency. Let $P = (X_1, Y_1, Z_1) \in E'(\mathbb{F}_{p^2})$ and $Q = (X, Y, 1)$, i.e. the affine representation can be used. The mixed addition and the line are computed as follows:

$$l_{P+Q}(S) = (Y_1 - Y_2Z_1)X_2 - (Y_1 - Y_2Z_1) \cdot x_s + (X_1 - X_2Z_1) \cdot y_s - (X_1 - X_2Z_1)Y_2$$

For more details, see [15].

```
input : $P \in G_1$, $Q \in G_2$, $r = |6u+2| = \sum_{i=0}^{\log_2(r)} r_i 2^i$
output: $f_{|r|,Q}(P) \cdot l_{[-|r|]Q,\pi_p(Q)}(P) \cdot l_{-|r|Q+\pi_p(Q),-\pi_p^2(Q)}(P)$
1  $T \leftarrow Q$, $f \leftarrow 1$
2  for $i = \lfloor \log_2(r) \rfloor - 1$ to 0 do
3      $f \leftarrow f^2 \cdot l_{T,T}(P)$, $T \leftarrow 2T$
4      if $r_i = 1$ then
5          $f \leftarrow f \cdot l_{T,Q}(P)$, $T \leftarrow T + Q$
6      end
7  end
8  $Q_1 \leftarrow \pi_p(Q)$, $Q_2 \leftarrow \pi_p^2(Q)$
9  if $u < 0$ then
10     $T \leftarrow -T$, $f \leftarrow f^{-1}$
11 end
12 $f \leftarrow f \cdot l_{T,Q_1}(P)$, $T \leftarrow T + Q_1$
13 $f \leftarrow f \cdot l_{T,-Q_2}(P)$, $T \leftarrow T - Q_2$
14 return $f$
```

Algorithm 5: Miller's (revised) algorithm used in this work.

4.3 Inversion Trick

The Miller function for optimal-ate pairing is defined as $a_{opt} = f_{r,Q}(P) \cdot l_{[r]Q,\pi_p(Q)}(P) \cdot l_{[r]Q+\pi_p(Q),-\pi_p^2(Q)}(P)$, where $\pi_p(Q)$ is the Frobenius endomorphism and $r = |6u+2|$. As the parameter u is negative, resulting into $r < 0$, a special treatment is required. A cheap negation in G_2 is required to compensate the terms with $[-|r|]Q$. As $f_{-|r|,Q}(P) = (f_{|r|,Q}(P))^{-1}$, an expensive inversion in $G_T = \mathbb{F}_{p^k}^*/(\mathbb{F}_{p^k}^*)^r$ is also needed.[1]

The inversion trick by Aranha *et al.* [1] allows to replace the expensive inversion with a simple conjugation. Let $a_{opt}(Q,P) = [g^{-1} \cdot h]^{(p^{12}-1)/12}$, where $g = f_{|r|,Q}(P)$ and $h = l_{[-|r|]Q,\pi_p(Q)}(P) \cdot l_{-|r|Q+\pi_p(Q),-\pi_p^2(Q)}(P)$. Factorizing the power, we obtain

$$\begin{aligned}(g^{-1} \cdot h)^{\frac{p^{12}-1}{n}} &= g^{\frac{1-p^{12}}{n}} \cdot h^{\frac{p^{12}-1}{n}} = g^{\frac{(1+p^6)(1-p^6)}{n}} \cdot h^{\frac{p^{12}-1}{n}} = g^{\frac{(1+p^6)(p^{12}-p^6)}{n}} \cdot h^{\frac{p^{12}-1}{n}} \\ &= g^{p^6} g^{\frac{(1+p^6)(p^6-1)}{n}} \cdot h^{\frac{p^{12}-1}{n}} = g^{p^6(p^{12}-1)/n} \cdot h^{\frac{p^{12}-1}{n}} \\ &= (g^{p^6} \cdot h)^{\frac{p^{12}-1}{n}}.\end{aligned}$$

The expensive inversion is thereby replaced with an exponentiation by p^6, which is equivalent to one conjugation in the corresponding group (details to follow). The conjugation costs 3 negations in $\mathbb{F}_{p^2}$.

4.4 Final Exponentiation

Recalling from section 3.5.2, the final exponentiation process provides the unique result of the pairing. In case of embedding degree $k = 12$, the goal is to compute $f^{(p^{12}-1)/n}$, where $f \in \mathbb{F}^*_{p^{12}}/(\mathbb{F}^*_{p^{12}})^n$ is the output of the Miller's loop. The exponential can be factored as $(p^{12}-1)/r = (p^6-1)\cdot(p^2+1)\cdot(p^4-p^2+1)/r$, dividing it into two distinct computation processes.

The computation of $f^{(p^6-1)(p^2+1)}$ is the easy part of the final exponentiation. The expression corresponds to $(\bar{f}\cdot f^{-1})^{p^2+1}$, which costs a cheap conjugation, a field inversion, two multiplications, and one Frobenius mapping in the field $\mathbb{F}_{p^{12}}$.[10]

For the hard part, i.e. $f^{(p^4-p^2+1)/n}$, a slightly optimized version of the Fuentes-Castaneda exponentiation [20] is used. Again, recalling the parametrization for BN curves: $p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$ and $n(u) = 36u^4 + 36u^3 + 18u^2 + 6u + 1$. The hard exponent can now be rewritten as a polynomial in respect to the parameter u as follows:

$$\begin{aligned}\frac{p(u)^4 - p(u)^2 + 1}{n(u)} =& -36u^3 - 30u^2 - 18u - 2\\ &+ p(u)(-36u^3 - 18u^2 - 12u + 1)\\ &+ p(u)^2(6u^2 + 1)\\ &+ p(u)^3.\end{aligned}$$

To proceed, the following chain of operations is first calculated:

$$f^u \rightarrow f^{2u} \rightarrow f^{4u} \rightarrow f^{6u} \rightarrow f^{6u^2} \rightarrow f^{12u^2} \rightarrow f^{12u^3}$$

Then, let $a = f^{6u}\cdot f^{6u^2}\cdot f^{12u^3}$ and $b = a\cdot(f^{2u}\cdot f)^{-1}$. The final result of the pairing is obtained as

$$f = [f^{6u^2}\cdot f\cdot f^p\cdot a][b]^p[a]^{p^2}[b]^{p^3}.$$

The proposed method requires altogether 3 exponentiations by u, 3 squarings and 11 multiplications of $\mathbb{F}_{p^{12}}$, which is one multiplication more than in the original method by Fuentes-Castaneda[1]. As an exchange, three large temporary variables of $\mathbb{F}_{p^{12}}$ are only required, instead of four, reducing the amount of needed RAM.[37]

[1]Relatively inexpensive Frobenius operations and inversions are ignored.

4.4.1 Cyclotomic Subgroups

The easy part of final exponentiation has an important consequence. After the exponentiation of the easy part, the element f becomes a member of a *cyclotomic subgroup* $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2}) = \{\alpha \in \mathbb{F}_{p^k} \mid \alpha^{\Phi_{12}(p)} = 1\}$, where $\Phi_{12}(p) = p^4 - p^2 + 1$. Rewriting the easy part, one obtains

$$(p^6 - 1)(p^2 + 1) = (p^6 - 1)\frac{p^6 + 1}{p^4 - p^2 + 1} = \frac{p^{12} - 1}{\Phi_{12}(p)}.$$

It follows that $(f^{(p^6-1)(p^2+1)})^{\Phi_{12}(p)} = f^{p^{12}-1} = 1$, and thus $f \in \mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$.

The elements of cyclotomic subgroups have several beneficial properties; they especially enable efficient formulas for squaring. In cyclotomic squaring formulas by Granger and Scott [21], the elements are compressed during the squaring and then decompressed. The method especially suits for the square-and-multiply process, as squaring can be done sequentially without decompressing. It is especially useful as the hard part requires costly exponentiations by the 128-bit parameter u.

Chapter 5

Implementation and Results

The implementation of this thesis builds on the library provided by Unterluggauer *et al.* [37], which targets the 32-bit Cortex-M0+ architecture. The proposed library was built and analyzed on the nRF9160 development kit [32] equipped with the ARM Cortex-M33 [3] microprocessor. The goal was to optimize the cycle consumption and run time of the O-ate pairing for the given design and analyze the feasibility of the usage in practice.

The use of a more capable, backwards binary compatible M33-architecture enabled an optimization on the multiplication, giving a significant speedup. Moreover, allocating the heavy and frequently used routines on the RAM gives a slight improvement in the performance.

In the first section, an overview of the used library is given. The second section presents the setup used in this implementation; the pairing parameters and the used hardware. The third section presents the results along with a brief comparison. Fourth chapter contains the discussion, analyzing the results, the situation of pairings in research and further improvements. Fifth section covers the learning outcome of the thesis.

5.1 Overview of the Library

The library[1] is written in C and supported with optimized assembly routines for integer arithmetic. The target platform is 32-bit ARM Cortex-M0+. The Cortex-M microprocessor family is optimized for constrained embedded platforms. The library

[1]Available in https://github.com/IAIK/pairings_in_c.

provides a comprehensive set of routines for both pairing and elliptic curve-based cryptography for Cortex-M based embedded platforms. It provides functions for the O-ate pairing, as well as for multiplication and division of the pairing functions. The provided curves are BN254[2] and BN158. The arithmetic is layered, as described in chapter 4. The library also provides several pairing-based protocols, for example, the Identity Based Encryption (IBE) [11] protocol, as well as integrated tools for benchmarking and analyzing.

The hardware requirements of the library are low; considerably below 100kGE[3], which is required in many related libraries. The library is designed to be applied for interactive protocols, feasible for embedded applications, such as wireless sensor nodes.[37]

Three different hardware architectures are supported: a plain microprocessor platform, a microprocessor with a multiply-accumulate instruction-set extension, and a CPU with a dedicated hardware accelerator. The designated hardware, such as a drop-in module or MAC instruction set extension provide significant speedup but require additional hardware development. The drop-in hardware utilizes separate arithmetic units for field computations and is memoryless and requires neither multi-master bus nor direct memory access.[37]

The prime field arithmetic is realized with SPS (Separated Product Scanning) variant of Montgomery multiplication. The reduction step is always performed after the prime-field operation to ensure constant runtime, which provides side-channel security.[37]

5.2 Testbench

For this implementation, we chose the plain microprocessor design. The BN254 curve was chosen i.e. the parameter $u = -\texttt{40800000 00000001}_{\texttt{h}}$. The used elliptic curve is $y^2 = x^3 + 2$, embedding degree $k = 12$ and the tower extensions as presented in 4.1.1. The 64-bit parameter u satisfies $\log_2 n(u) \leq 256$ and $3000 \leq k \cdot \log_2 p(u) \leq 5000$ which, by the time of the article [10], correspond to a security level of 128-bit AES. However, after the recent study [6], the security of BN254 fell to roughly 100 bits. The option of BN158 was ignored, as it can be considered insecure.

As mentioned, the used platform was nRF9160 cellular IoT development kit (DK), which contains 64 MHz ARM Cortex-M33 CPU, 1 MB flash and 256 KB RAM. The experiment was carried out on actual hardware. It contains manifold connectivity;

[2]Where $u = -2^{62} - 2^{55} - 1$ as described earlier

[3]GE refers to *Gate-Equivalents*, which is a measure for the complexity of a digital circuit

bluetooth, LTE-M/NB-IoT modem and GPS.[32]

The software uploaded to the DK was a minimalistic, bare-metal solution. It consists of the CMSIS (Cortex Microcontroller Software Interface Standard) (version 5.7.0), nRF SDK (Software Development Kit) (version 16.0.0) and the pairing library with the integrated benchmark tool. The CMSIS is the HAL (Hardware Abstraction Layer) for ARM Cortex based microcontrollers, providing the generic tool interfaces for interacting with the CPU. nRF SDK in turn provides the firmware functionality for the nRF on-board hardware, such as UART and memory controlling. The software was compiled with the bare metal ARM gnu toolchain[4], with the `O3` optimization level.

```
.macro mulacc
  @ Fetch the least and most significant 16-bits of the input values.
  uxth r6, r1
  uxth r7, r2
  lsr  r1, r1, #16
  lsr  r2, r2, #16

  mov r0, r6
  mul r0, r0, r7  @ low * low
  mul r6, r6, r2  @ low * high
  mul r2, r2, r1  @ high * high
  mul r1, r1, r7  @ high * low

  lsl r7, r6, #16 @ Add the least significant 32-bit values
  lsr r6, r6, #16
  add r0, r7, r0
  adc r2, r6, r2

  lsl r7, r1, #16 @ Add the most significant 32-bit values
  lsr r6, r1, #16
  add r0, r7, r0
  adc r2, r6, r2

  mov r7, #0
  @ Set the carry values to be added to the next output element
  add r5, r5, r0
  adc r4, r4, r2
  adc r3, r3, r7 @ Clear the carry-flag
.endm
```

Listing 5.1: The original multiply-accumulate unit for M0 design.

5.2.1 Optimizations

Two optimizations on the hardware level were conducted to accelerate the pairing computation. Firstly, the big-integer multiplication was accelerated owing to the extended instruction set of the used M-33. Secondly, the code of two frequently used functions were allocated to the RAM to decrease the cycle consumption of the memory fetch. The optimizations are presented briefly in the following.

The assembler optimizations are designed for Cortex-M0 architecture which supports $32 \times 32 \rightarrow 32$ multiplication. Therefore, the code needs to split the multiplication into several parts to obtain the 64-bit output of two word-length integers. The multiplication and reduction with 128-bit values are carried out as numerous 32-bit multiplications that are chained in a multiply-accumulate fashion. The original multiply-accumulate macro is depicted in Listing 5.1.

The Cortex-M33, in turn, is capable of performing direct $32 \times 32 \rightarrow 64$ multiplication. As a result, the four multiplications and several shifts and additions could be replaced with a single instruction `umull`. The resulting macro is depicted in the Listing 5.2. As a result, the cycle consumption of the Karatsuba multiplication and Montgomery reduction algorithms decreased significantly. For instance, the cycle consumption of Karatsuba multiplier reduced from 2380 cycles to 1400 cycles, which is around 41% improvement.

```
.macro mulacc
  umull r6, r0, r1, r2

  mov r7, #0
  @ Carry values to be added to the next output element
  add r5, r5, r6
  adc r4, r4, r0
  adc r3, r3, r7 @ Clear the carry flag
.endm
```

Listing 5.2: Optimized multiply accumulate for m33

Initially, we aimed to utilize instruction cache as-is for a better performance. The instruction cache is a fast memory between the processor and the main non-volatile flash, allowing the CPU to execute code without wait states. Naturally, loading the cache from flash has a delay, and thus instruction cache should be loaded with code, that is used frequently. The instruction cache of nRF9160 has a size of 2048 bytes.[32] However, after enabling it we observed no improvement in the cycle consumption, and by profiling the memory usage, there appeared to be numerous cache read misses. It appeared that the used algorithms exceeded the size of the cache, and the remaining instructions had to be fetched from the FLASH, causing delay.

Therefore, we instead allocated two weighty and frequently used functions to the RAM, which also enables instruction fetching without wait states. We placed the main multiplier — 256-bit Karatsuba and the Montgomery reduction algorithms. The algorithms take 3,170 bytes of RAM in total, which is under 5% of the total RAM in nRF9160. As a result, the pairing computation decreased by 4 MCycles, which is roughly 12% improvement.

5.3 Results

The improvements in the key operations of pairing are depicted in Table 5.1. The table describes the cycle consumption in (i) the library built as-is[4], (ii) with the optimized multiplication, (iii) after the RAM-allocation, and (iv) the final build, containing the compiler optimizations. The percentages below present the total improvement gained from optimizations. The squaring in G_T corresponds to the efficient formulas in cyclotomic subgroups.

Table 5.2 presents the cycle consumption and run time of the additional operations utilized in many pairing-based cryptosystems, in the final setup. It contains the cycle cost of a key-encapsulation protocol which is based on the Boneh-Boyen ID-based encryption (as described in section 3.3). It gives an estimate of how costly an actual actual cryptographic action might be[5].

Table 5.1: The effect of optimizations in various arithmetic operations

	$\mathbb{F}_p$			G_T		$e(P,Q)$
	Add.	**Mul.**	**Inv.**	**Sqr.**	**Mul.**	**Pairing**
Optimization	(Cycles)	(Cycles)	(kCycles)	(kCycles)	(kCycles)	(kCycles)
Initial setup	260	4,580	1,262	144	224	50,768
Optimized mul.	270	2,770	723	93	141	32,006
RAM allocation	254	2,270	595	80	120	27,211
Final	**217**	**2,190**	**552**	**76**	**115**	**25,917**
Improvement	**16.5%**	**52.5%**	**56.3%**	**47.2%**	**48.7%**	**48.9%**

[4]Without any compiler optimizations

[5]The referred Boneh-Boyen ID-based encryption is considered impractical for actual implementations due to its complexity

Table 5.2: Time and cycle consumption of various pairing-related operations

	G_1	G_2	G_T	$e(P,Q)$		ID. based protocol	
	Mul.	**Mul.**	**Exp.**	**Mul.**	**Div.**	**Encaps.**	**Decaps.**
kCycles	**8,086**	**19,367**	**39,189**	**35,471**	**40,998**	**63,964**	**35,581**
Time (ms)	**131**	**310**	**623**	**565**	**651**	**1,012**	**567**

Table 5.3 presents the performance results of a single O-ate pairing and the memory consumption of the pairing library. The identity-based encryption interface is not counted in the memory mapping, as we only aimed to analyze the key functions for O-ate pairing, and the pairing-based protocols. RAM memory contains the allocated functions and the maximum stack usage within the library functions.

Table 5.3: Performance results of the pairing library

Optimal ate pairing				
kCycles	**Cache misses (%)**	**Runtime(ms)**	**RAM (Byte)**	**Flash (Byte)**
25,917	**0.45**	**410**	**5,386**	**15,946**

5.3.1 Comparison and Analysis

Table 5.4 presents our results among a few other related ate pairing implementations over BN254 curves: a smartcard design by Devegili *et al.* [17], the underlying library we utilized, and a Rasperry Pi3 solution by Hajny *et al.* [22]. All the implementations are embedded "plain microprocessor" designs.

Table 5.4: Embedded platform designs for optimal ate pairing over BN254 curve

		CPU Freq.	**Runtime**		**RAM**	**Flash**
Work	**Platform**	(MHz)	(kCycles)	(ms)	(Byte)	
[17]	Philips HiPerSmart	36	90,462	2,513	<16,000	-
[37]	Cortex-M0+	48	47,643	993	2,828	18,116
Ours	**nRF9160, Cortex-M33**	**64**	**25,917**	**410**	**5,386**	**15,946**
[22]	Rasberry Pi3, Cortex-A53	1200	-	13	-	-

The runtime decreased by over 500ms comparing to the original work. The runtime is likely decreased by a factor due to the higher clock frequency of the M33. The most significant benefit arises from the utilized single-instruction 32-bit multiplication. On the other hand, the extended multiplication also increases the amount of required gate-equivalents. The RAM usage is slightly larger but certainly not a bottle-neck. The Rasberry Pi3 solution is not directly comparable, due to the clock frequency and bigger caches, but it is still a good reference. Also, the power consumption of the Cortex-A family greatly exceeds the M-family.

The question of whether the pairing implementation in this form is feasible depends on the used protocol and how frequently the pairings are required to be used. The recent work of Rezaeibagha *et al.* [34] provides a secure and, by the time, the most efficient ID-based signature scheme with a cost of one exponentiation, i.e. scalar multiplication in G for signature generation, and two pairings for verification[6]. Within this setup, assuming that $G = G_1$, the signing and verifying would require 131 ms and $2 \cdot 410 = 820$ ms, correspondingly. Even with all the resources available, the presented setup could perform only a few signatures or one verification per second. Considering a network of multiple nodes requiring high integrity, authenticity, and non-repudiation between actions, this doesn't suffice. However, if an identity-based signature is used only to authenticate when establishing a connection, the present delay should be in sensible bounds. Such a situation could be a single sensor node joining a sensor network, and identifying itself to the base station, e.g. a server. A digital "handshake" of few seconds in the beginning of connection seems reasonable.

5.4 Discussion

The question of feasibility of our implementation is ambiguous — it depends on the use-case. The current setup is not capable of highly interactive protocols, such as digitally signing each frequently occurring action. For a more passive use, such as a single ID-based authentication, the setup suffices. Such a situation could be a device identifying itself to a server, which in turn has computational resources to verify multiple signatures in a second.

The further improvement of our work could include the further optimizing of the assembler routines. The ASM-optimizations of the utilized library seem to contain mostly unrolled code, leading to larger-size algorithms. The nRF9160 might benefit from looped structures, as they could fit better in the 2kB instruction cache. Also, the M33 supports direct multiply-accumulate instruction to 64-bit output, which could save some additions. Unfortunately, the instruction did not directly suit into the pre-existing library as-is.

The trend for embedded, constrained pairing implementations, from 2015 onwards seems to favor the usage of FPGA solutions or other hardware extensions. For instance,

[6]Excluding the setup- phase and parameter generation

the work of Sghaier *et al.* [36] provides a 225 MHz Virtex-6 FPGA design for pairings, capable of computing the O-ate pairing in 350 μs. In [2], the pairing is performed on multiple platforms with 1.0-1.7 GHz ARM CPUs, having the run time between 3.4-55 μs. A versatile pairing-based protocol would require either a sole high-frequency CPU or customized drop-in hardware. The higher frequency CPU would be more energy-consuming but easily deployed, whereas the FPGA solution requires a lot of development time.

Another considerable issue is the used Barreto-Naehrig curve family. An improvement found in Number Field Sieve (NFS), which computes DLP in $\mathbb{F}_{p^k}$, lowered the pairing extension field security such that BN curves with 256-bit primes $p(u)$ correspond slightly under 100 bits of security, instead of 128.[6] For 128 bits of security, a curve with 384-bit prime $p(u)$, for example, BN384, should be used. Moreover, the novel Barreto-Lynn-Scott (BLS) curves seem to offer significantly better performance than BN curves, with the same security level. In fact, many research suggests that BLS12[7] is the most efficient (known) curve to compute the pairing. In [6], at the 128-bit security level, the optimal ate pairing with BN curves costs $17774m + 4i$, whereas with BLS12, it costs $14028m + 6i$.

However, neither the code optimizations nor enhanced methodology for pairings would significantly increase the performance of this work; the pairing would still require a few hundred milliseconds. A hardware improvement would certainly be needed to have the capabilities for practical pairing-solutions.

[7]Curve over the extension field, with $k = 12$

Chapter 6

Conclusions

In this work, optimal ate pairing over the BN254 curve is implemented on the nRF9160 platform, endowed with a single 64 Mhz ARM Cortex-m33 microprocessor. An open-source library is used, with optimized assembly routines to support the 64-bit multiplication of the m33 instruction set. As a result, the optimal ate pairing is effectively computed in 410 ms.[1] In practice, the resulting implementation is applicable for infrequent use, such as in the authentication process. The work shows that the full-extent pairing-based cryptography for single-microprocessor ultra-low-power devices is yet out of reach. A practical approach requires to include a hardware-accelerator, for example, a customized FPGA. However, the devices with over 1.0 GHz microprocessors seem to be capable of performing the pairing-based cryptography.

The pairing-based cryptography overall seems to be in a phase of continuous development. The computation of Miller's algorithm and final exponentiation constantly take small improvement steps. The optimal ate pairing over BLS and BN curves seems to have been established as the main tool for pairing-based cryptography. Many open-source libraries exist for pairings, but it has not yet found its place as the part of larger cryptography libraries, such as the OpenSSL or MBEdtls. Despite the improvements, pairing-based cryptography will likely remain computationally heavy. The pairing-based cryptosystems are also developed at an accelerating pace, getting more and more optimized, as can be seen in comparison by [34]. Also, there doesn't seem to be concern about their security; the basis is in the BDH-problem, and DLP over prime and extension fields which have withstood for many years of security research.

The research on bilinear-pairings is clearly important, both in the general and the embedded platform framework. The novel protocols can enable innovative and more

[1]Version available in https://github.com/mhspes/pairings_in_c.

secure applications and might be a good response to the security needs for future IoT applications, such as sensor networks, autonomous systems, and many other applications. The current research has already been promising but the pairings still need few more years to stabilize its place in the cryptographic scene.

Bibliography

[1] Aranha, D., Karabina, K., Longa, P., Gebotys, C., López, J., & Paterson, K. (2011). Faster Explicit Formulas for Computing Pairings over Ordinary Curves. In Advances in Cryptology – EUROCRYPT 2011: 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings (Vol. 6632, pp. 48–68). https://doi.org/10.1007/978-3-642-20465-4_5

[2] Azarderakhsh, R., Fishbein, D., Grewal, G., Hu, S., Jao, D., Longa, P., & Verma, R. (2017). Fast Software Implementations of Bilinear Pairings. IEEE Transactions on Dependable and Secure Computing, 14(6), 605–619. https://doi.org/10.1109/TDSC.2015.2507120

[3] Arm Ltd. (n.d.). Cortex-M33 – Arm. Arm | The Architecture for the Digital World. Retrieved August 23, 2020, from https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m33

[4] Arm Ltd. (n.d.-b). GNU Toolchain | GNU Arm Embedded Toolchain. Arm Developer. Retrieved September 10, 2020, from https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm

[5] Balasubramanian, R., & Koblitz, N. (1998). The Improbability That an Elliptic Curve Has Subexponential Discrete Log Problem under the Menezes—Okamoto—Vanstone Algorithm. Journal of Cryptology, 11(2), 141–145. https://doi.org/10.1007/s001459900040

[6] Barbulescu, R., & Duquesne, S. (2018). Updating Key Size Estimations for Pairings. Journal of Cryptology, 32(4), 1298–1336. https://doi.org/10.1007/s00145-018-9280-5

[7] Barker, E. B. (2013). *Digital Signature Standard (DSS)* (No. Federal Inf. Process. Stds.(NIST FIPS)-186-4).

[8] Barreto, P., & Naehrig, M. (2006). *Pairing-friendly elliptic curves of prime order.* Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 3897, 319–331. https://doi.org/10.1007/11693383_22

[9] Bernstein, D. (2006). Curve25519: New Diffie-Hellman speed records. Public Key Cryptography - Pkc 2006, Proceedings, 3958, 207–228.

[10] Beuchat, J., González-Díaz, J., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T., ... Otsuka, A. (2010). High-Speed Software Implementation of the Optimal Ate Pairing over Barreto–Naehrig Curves. In Pairing-Based Cryptography - Pairing 2010: 4th International Conference, Yamanaka Hot Spring, Japan, December 2010. Proceedings (Vol. 6487, pp. 21–39). https://doi.org/10.1007/978-3-642-17455-1_2

[11] Boneh, D., & Boyen, X. (2004). Secure identity based encryption without random oracles. Advances In Cryptology - Crypto 2004, Proceedings, 3152, 443–459.

[12] Boneh, D., & Franklin, M. (2001). Identity-based encryption from the weil pairing. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2139, 213–229.

[13] Certicom Research: Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0. (2010). Retrieved June 17, 2020, from http://www.secg.com.

[14] Costello, C., Hisil, H., Boyd, C., Gonzalez Nieto, J., & Wong, K. (2009). Faster pairings on special weierstrass curves. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5671, 89–101. https://doi.org/10.1007/978-3-642-03298-1_7

[15] Costello, C., Lange, T., & Naehrig, M. (2010). Faster pairing computations on curves with high-degree twists. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6056, 224–242. https://doi.org/10.1007/978-3-642-13013-7_14

[16] Devegili, A.J.,Héigeartaigh, C.O., Scott, M., & Dagab, R. (2006). Multiplication and Squaring on Pairing-Friendly Fields. Cryptology ePrint Archive, Report 2006/471.

[17] Devegili, A., Scott, M., & Dahab, R. (2007). Implementing Cryptographic Pairings over Barreto-Naehrig Curves. In Pairing-Based Cryptogra-

phy – Pairing 2007 (Vol. 4575, pp. 197–207). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-73489-5_10

[18] D. Jao. (2010). *Elliptic curve cryptography.* http://goo.gl/wwN5aJ.

[19] Fagen Li, & Pan Xiong. (2013). Practical Secure Communication for Integrating Wireless Sensor Networks Into the Internet of Things. IEEE Sensors Journal, 13(10), 3677–3684. https://doi.org/10.1109/JSEN.2013.2262271

[20] Fuentes-Castañeda, L., Knapp, E., & Rodríguez-Henríquez, F. (2012). Faster Hashing to G2. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7118, 412–430. https://doi.org/10.1007/978-3-642-28496-0_25

[21] Granger, R., & Scott, M. (2010). Faster squaring in the cyclotomic subgroup of sixth degree extensions. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6056, 209–223. https://doi.org/10.1007/978-3-642-13013-7_13

[22] Hajny, J., Dzurenda, P., Ricci, S., Malina, L., & Vrba, K. (2018). Performance Analysis of Pairing-Based Elliptic Curve Cryptography on Constrained Devices. 2018 10th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2018-, 1–5. https://doi.org/10.1109/ICUMT.2018.8631228

[23] Hankerson, D., Vanstone, S., & Menezes, A. (2003). Guide to elliptic curve cryptography (pp. 6, 17-19, 26-29, 76-80, 153-160, 184-186). New York: Springer.

[24] Koblitz, N. (1987). Elliptic curve cryptosystems. Mathematics of Computation, 48(177), 203–209. https://doi.org/10.1090/S0025-5718-1987-0866109-5

[25] Lounis, A., Hadjidj, A., Bouabdallah, A., & Challal, Y. (2012). Secure and Scalable Cloud-Based Architecture for e-Health Wireless Sensor Networks. 1–7. https://doi.org/10.1109/ICCCN.2012.6289252

[26] Lu, Y., & Li, J. (2019). Constructing designated server public key encryption with keyword search schemes withstanding keyword guessing attacks. International Journal of Communication Systems, 32(3), e3862–n/a. https://doi.org/10.1002/dac.3862

[27] Menezes, A., Okamoto, T., & Vanstone, S. (1993). Reducing elliptic curve logarithms to logarithms in a finite field. IEEE Transactions on Information Theory, 39(5), 1639–1646. https://doi.org/10.1109/18.259647

[28] Miller, V. (2004). The Weil Pairing, and Its Efficient Calculation. Journal of Cryptology, 17(4), 235–261. https://doi.org/10.1007/s00145-004-0315-8

[29] Miller, V. (1986). Use of Elliptic Curves in Cryptography. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 218, 417–426. https://doi.org/10.1007/3-540-39799-X_31

[30] Nadia El Mrabet, N., & Joye, M. (2017). Guide to Pairing-Based Cryptography. Chapman and Hall/CRC. https://doi.org/10.1201/9781315370170

[31] Naehrig, M., Niederhagen, R., & Schwabe, P. (2010). New software speed records for cryptographic pairings. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6212, 109–123. https://doi.org/10.1007/978-3-642-14712-8_7

[32] nRF9160 Product Specification v2.0. Nordic Semiconductor. (2020). https://infocenter.nordicsemi.com/pdf/nRF9160_PS_v2.0.pdf

[33] Paar, C., & Pelzl, J. (2010). Understanding Cryptography: A Textbook for Students and Practitioners. (pp. 3-5, 152-155, 216-218, 241-250). https://doi.org/10.1007/978-3-642-04101-3

[34] Rezaeibagha, F., Mu, Y., Huang, X., Yang, W., & Huang, K. (2019). Fully Secure Lightweight Certificateless Signature Scheme for IIoT. IEEE Access, 7, 144433–144443. https://doi.org/10.1109/access.2019.2944631

[35] Schoof, R. (1995). Counting points on elliptic curves over finite fields. Journal De Théorie Des Nombres De Bordeaux, 7(1), 219-254. doi: 10.5802/jtnb.142

[36] Sghaier, A., Zeghid, M., Ghammam, L., Duquesne, S., Machhout, M., & Ahmed, H. (2018). High speed and efficient area optimal ate pairing processor implementation over BN and BLS12 curves on FPGA. Microprocessors and Microsystems, 61, 227–241. https://doi.org/10.1016/j.micpro.2018.06.001

[37] Unterluggauer, T., & Wenger, E. (2014). *Efficient pairings and ECC for embedded systems.* Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8731, 298–315.

[38] Vercauteren, F. (2010). Optimal Pairings. IEEE Transactions on Information Theory, 56(1), 455–461. https://doi.org/10.1109/TIT.2009.2034881

[39] Washington, L. (2008). *Elliptic curves: Number theory and cryptography* (pp. 9-15, 77, 93-95, 130, 151-159, 170-171, 480-481). Boca Raton, FL: Chapman & Hall/CRC.