Technion - Israel Institute of Technology
Project in Computer Security−Elliptic Curve Cryptography

October 2014

Pairing-based Short Signatures

Amit Markel amit@markel.co
Leonid Nemirovskiy krusnik013@gmail.com

Barukh Ziv *(Supervisor)*
barukh.ziv@gmail.com

Abstract

Recent development in research and practical use of elliptic curves in public key cryptography motivates us to investigate the field: the discrete logarithm problem on elliptic curves is yet to be solved in sub exponential time, we benefit from the same level of security such as one would achieve using RSA whilst using substantially smaller key sizes and digital signatures, noticeably reducing expensive network traffic load in terms of power and transmission time [7], as well as storage size requirements in favor of computation complexity.

Contents

Part I
Introduction

Cryptography is an important field in computer science, and digital signatures in this field are prominent. We investigate different parameters of the digital signature in quest of improvement. One of the most important vectors in this pursuit is the signature length, on which we focus our attention.

The Diffie-Hellman problems play a major role in attaining shorter signature lengths as well as appropriate pairing maps from elliptic curve point groups to multiplicative ordinary number groups. Shorter signatures require more complex underlying field size, in particular - extension fields of higher order; this allows reach of similar security parameters to omnipresent digital signature standard, in trade of faster verification computation time for instance.

We use elliptic curves of two compromising types: one of *lesser security* parameter which provides *better speed* (approximates to 1024-bit RSA), and another kind which delivers *better security* (approximates to 2048-bit RSA).

We compare differing pairing maps such as The Weil Pairing and The Tate Pairing, by implementing and optimizing these with various techniques - we developed a well optimized algorithm for Tate, Simplified Tate Pairing, using elliptic nets - achieving better results than in some articles ([6]) which utilize the elliptic nets idea.

We tested two short signature schemes: BLS [13] and our modified scheme of ZSS [14], experiencing better performance with ZSS after supreme optimizations made.

We implemented a library in `C++` that allows one to perform elliptic curve point arithmetics, pairing maps calculations, signature schemes generation and verification, and various other useful utilities – its main highlight is its simplicity: friendly user interfaces in terms of usage and comprehension.

Our signatures (160 bits) are twice as short as ECDSA standard signatures (320 bits), for security level of 80 bits. (1024 bits for RSA).

Part II
Theory

1 Elliptic curves

Definition 1. An elliptic curve over a finite field $\mathrm{GF}(q^\alpha)$ for a prime q and appropriate $\alpha > 0$, is a set of points which satisfy the Weierstrass equation

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

where $a, b \in \mathrm{GF}(q^\alpha)$ for which $4a^3 + 27b^2 \neq 0$. This set includes a point at infinity, denoted by $\mathcal{O}$.

The number of points on E (including $\mathcal{O}$) is called the *order* of E.

Notation. We mark sets of points on the curve E over $\mathrm{GF}(q^\alpha)$ as $E(q^\alpha)$, and its *order* by $\#E(q^\alpha)$.

Remark. One may observe that: if $a, b \in \mathrm{GF}(q)$ then $a, b \in \mathrm{GF}(q^\alpha)$, thus $E(q) \subseteq E(q^\alpha)$, allowing usage of the same points (and hence the same curve) over the extension field.

Example. For a curve E over $\mathrm{GF}(13)$ as $y^2 = x^3 + 10x + 5$, one can find

$$\begin{aligned} E(13) &= \{\mathcal{O}, (1,4), (1,9), (3,6), (3,7), (8,5), (8,8), (10,0), (11,4), (11,9)\}, \\ \#E(13) &= 10. \end{aligned}$$

1.1 Operations

Theorem. *$G = E(q^\alpha)$ is an addition group with the following operations.*
(Let $G^+ = G \setminus \{\mathcal{O}\}$)

1. *An inverse of a point $\mathbf{P} = (x, y) \in G^+$ is $-\mathbf{P} = (x, -y)$.*
Remark. $-\mathcal{O} = \mathcal{O}$.

2. *Let $\mathbf{P}, \mathbf{Q} \in G^+$, then $\mathbf{R} \coloneqq \mathbf{P} + \mathbf{Q} \in G^+$ where $\mathbf{P}, \mathbf{Q}, -\mathbf{R}$ lie on a common line.*
Remark. $\mathbf{P} + \mathcal{O} = \mathbf{P}$, $\mathbf{P} + (-\mathbf{P}) = \mathcal{O}$.

Definition 2. A scalar multiplication of a point $\mathbf{P} \in E(q^\alpha)$ is defined by

$$[n]\mathbf{P} = \underbrace{\mathbf{P} + \cdots + \mathbf{P}}_{n} \text{ for } n \in \mathbb{N}^+.$$

Remark. $[0]\mathbf{P} = \mathcal{O}$, $[-n]\mathbf{P} = [n](-\mathbf{P})$.

1.2 Algorithms and Optimizations

1.2.1 Adding and Doubling

One should distinguish between self adding of a point - doubling, and addition of two distinct points - adding.

The standard algorithms for point addition and doubling are defined in [1], we would like to focus on an optimized approach for these operations for our use.

Projective coordinates (*Jacobian coordinates*). Inversion operations are costly, therefore one may define points differently with an additional dimension in order to replace such heavy computations with other operations, as follows.

$$(x, y) \quad \mapsto \quad (X, Y, Z) \quad \text{s.t. } x = \frac{X}{Z^2},\ y = \frac{Y}{Z^3}.$$

Remark. Conversion from standard to projective coordinates is done by setting Z to 1.

Remark. The Jacobian coordinates for $\mathcal{O}$ are $\left(\delta^2, \delta^3, 0\right)$ for $\delta \in \mathrm{GF}\left(q^\alpha\right) \setminus \{0\}$.

Let $E:\ y^2 = x^3 + ax + b$ over $\mathrm{GF}\left(q^\alpha\right)$.

Doubling [1]. $[2]\left(X_1, Y_1, Z_1\right) = \left(\mathsf{X}_2, \mathsf{Y}_2, \mathsf{Z}_2\right)$ on E where

$$\begin{array}{rcl} M & \coloneqq & 3X_1^2 + aZ_1^4, \\ \mathsf{Z}_2 & = & 2Y_1Z_1, \\ S & \coloneqq & 4X_1Y_1^2, \\ \mathsf{X}_2 & = & M^2 - 2S, \\ T & \coloneqq & 8Y_1^4, \\ \mathsf{Y}_2 & = & M\left(S - X_2\right) - T. \end{array},$$

performing 10 field multiplications in total.

Optimization. For $a = 0$, one conducts less calculations when computing M: performing 8 field multiplications instead.

Adding (assuming distinct points) [1]. $\left(X_0, Y_0, Z_0\right) + \left(X_1, Y_1, Z_1\right) = \left(\mathsf{X}, \mathsf{Y}, \mathsf{Z}\right)$ on E where

$$\begin{array}{rcl} U_0 & \coloneqq & X_0Z_1^2, \\ S_0 & \coloneqq & Y_0Z_1^3, \\ U_1 & \coloneqq & X_1Z_0^2, \\ S_1 & \coloneqq & Y_1Z_0^3, \\ W & \coloneqq & U_0 - U_1, \\ R & \coloneqq & S_0 - S_1, \\ T & \coloneqq & U_0 + U_1, \\ M & \coloneqq & S_0 + S_1, \\ \mathsf{Z} & = & Z_0Z_1W, \\ \mathsf{X} & = & R^2 - TW^2, \\ V & \coloneqq & TW^2 - 2\mathsf{X}, \\ \mathsf{Y} & = & 2^{-1}\left(VR - MW^3\right). \end{array}$$

, performing 16 field multiplications total.

Optimization. For $Z_1 = 1$ (Standard coordinates for the second point), one may spare the two first computations for U_0 and S_0: performing 11 field multiplications instead.

1.2.2 Scalar multiplication

Algorithm 1 Simple binary algorithm.

Input. A positive integer n, and a *Point* $\mathbf{P}$.
Output. The *Point* $[n]\mathbf{P}$.

1. Let $b_k b_{k-1} \cdots b_1 b_0$ be the binary representation of n.
2. Set $\mathbf{Q} \leftarrow \mathcal{O}$.
3. For $j = k$ downto 0 by 1 do
   3.1 Set $\mathbf{Q} \leftarrow [2]\mathbf{Q}$,
   3.2 If $b_j = 1$ then Set $\mathbf{Q} \leftarrow \mathbf{Q} + \mathbf{P}$.
4. Output $\mathbf{Q}$.

Performing $\log_2 n$ point doublings and HammingWeight (n) point additions.

Optimization 1. We can compute less than HammingWeight (n) point additions, when using alternative representations of integers by using other bases such as a *non-redundant* signed m-ary representation with $\left\{-\left(2^{r-1}-1\right), ..., -1, 0, 1, ..., \left(2^{r-1}-1\right)\right\}$ as base for some small natural r i.e. $3, 4, 5$. We pay with a small constant number of field inversions, alas achieving a reduction of point additions. The resulting algorithm is similar to the binary case - moreover on this modification in [2].

Optimization 2. In various occasions we use the same point $\mathbf{P}$ for scalar multiplications, so we would like to precompute data in advance and modify the algorithm accordingly as follows. This optimization eliminates all point doublings for the fixed-point scalar multiplication.

Algorithm 2 Pre-computation.

Input. A *Point* $\mathbf{P}$ of order k.
Output. Precomputed data array for computing $[n]\mathbf{P}$ for $n \leq k$.

1. Set Array $\leftarrow$ *Array* of *Point*.
2. For $i = 0$ to $(1 + \log k)$ by 1 do
   2.1 Array add $\left[2^i\right]\mathbf{P}$.

Algorithm 3 Modified Binary algorithm with pre-computation.

Input. A *Point* $\mathbf{P}$, a positive integer $n \leq \text{order}(\mathbf{P})$, and a precomputed *Array*.
Output. The point $[n]\mathbf{P}$.

1. Let $b_k b_{k-1} \cdots b_1 b_0$ be the binary representation of n,
2. Set $\mathbf{Q} \leftarrow \mathcal{O}$.
3. For $j = 0$ to k by 1 do
   3.1 If $b_j = 1$ then Set $\mathbf{Q} \leftarrow \mathbf{Q} + \text{Array}[j]$.
4. Output $\mathbf{Q}$.

Optimization 3. Further improvement can bring to increased speedup gain, by using the *non-adjacent form* (NAF) representation instead of binary [2]. This representation extends the base into $B = \{-1, 0, 1\}$, yielding the conversion of a number of the form $\sum_{j=0}^{k} b_j 2^j$ where $b_j \in \{0, 1\}$, into one of the form $\sum_{j=0}^{k+1} s_j 2^j$ where $s_j \in B$ such that its hamming weight is minimal. Then, one can modify the previous algorithm, as we show next.

Algorithm 4 Modified NAF algorithm with pre-computation.

Input. A *Point* $\mathbf{P}$, a positive integer $n \leq \text{order}(\mathbf{P})$, and a precomputed *Array*.
Output. The *Point* $[n]\mathbf{P}$.

1. Let $s_{k+1} s_k s_{k-1} \cdots s_1 s_0$ be the *NAF* representation of n,
2. Set $\mathbf{Q} \leftarrow \mathcal{O}$.
3. For $j = 0$ to $k + 1$ by 1 do
   3.1 Select Case s_j
      i. Case $s_j = 1$: Set $\mathbf{Q} \leftarrow \mathbf{Q} + \text{Array}[j]$.
      ii. Case $s_j = -1$: Set $\mathbf{Q} \leftarrow \mathbf{Q} - \text{Array}[j]$.
4. Output $\mathbf{Q}$.

1.2.3 Finding a random point

Let $E: y^2 = x^3 + ax + b$ over GF (q^α).

Completely random points on the curve. Let $\mathsf{x} \in \text{GF}(q^\alpha)$, then one is able to find a value of y by substituting x with x on the right-hand side of the equation and computing its square root over the extension field [15].

Random points of large prime order p**.** Let $\mathbf{P} \in E(q^\alpha)$ where order $(P) = p$, then one may define an integer n of their choosing and output $[n]\mathbf{P}$.

Lemma. *A point* $\mathbf{P} \in E(q)$ *where order*(P) *is a large prime divisor of* $\#E(q)$ *may be found in the following way.*

Algorithm 5 Finding a random point of large prime order.

`Input.` `A large prime divisor` p `of` $\#E(q)$`.`
`Output.` `The` *Point* $\mathbf{P}$ `of order` p`.`

1. `Set` $\mathbf{P} \leftarrow$ `random` *Point*`.`
2. `If` $[p]\mathbf{P} = \mathcal{O}$ `then Output` $\mathbf{P}$`.`
3. `Set` $\mathbf{P} \leftarrow \left[\frac{\#E(q)}{p}\right]\mathbf{P}$`.`
4. `If` $\mathbf{P} = \mathcal{O}$ `then Goto 1.`
5. `Output` $\mathbf{P}$`.`

1.3 Complex Multiplication Method (CM)

1.3.1 Overview

CM is a method of constructing curves given a fundamental discriminant $-D$, a basic parameter to the procedure. A main portion of the process is finding the *Hilbert* class polynomial, defined by $H_D(x) = \prod(x - j(\alpha))$, where α runs over all complex numbers such that $(\alpha, 1)$ is a zero of one of the h_D (class number) inequivalent primitive reduced forms of discriminant $-D$. Intrigued readers should enquire into a book, such as [20], for the details. We will describe an algorithm shortly, in which we use to generate such a curve.

1.3.2 Generating curves using CM

We seek curves over GF(q) which contains j-invariant of such curves. We would like to obtain a discriminant $-D$ for which $H_D(x)$ has a root in GF(q). Such a prime number q is for which the diophantine equation

$$4q = x^2 + Dy^2 \tag{1}$$

can be solved.

Theorem 3. [2]

1. Given a solution (x, y, q) *to equation (1),* $m = q+1\pm x$ *is a possible number of points of curves over GF*(q) *with* j*-invariants which are roots of the Hilbert class polynomial* $H_D(x)$*.*

2. If $D > 4$ then all elliptic curves with given j-invariants (where $j \neq 0, 1728$) over $GF(q)$ are given by

$$y^2 = x^3 + 3kc^2x + 2kc^3$$

where $k = j/(1728 - j)$ and $c \in GF(q)$.

3. Let E, E' be curves with the same j-invariant but not isomorphic over $GF(q)$. Thus, if $j \neq 0, 1728$ then E' is a quadratic twist of E and if $\#E(q) = q + 1 - x$ then $\#E'(q) = q + 1 + x$.

4. Suppose $j \neq 0, 1728$. If $E: \ y^2 = x^3 + ax + b$ then E' can be given by $y^2 = x^3 + ac^2x + bc^3$ where c is any quadratic non-residue in $GF(q)$.

We may now describe an algorithm to generate a curve using CM.

Algorithm 6 Generate a curve using CM.

`No Input.`
`Output.` A curve E over GF(q) with acceptable $\#E(q)$.

1. Find a triple (x, y, q) solution to equation (1).
2. **Set** $m \leftarrow q + 1 + x$.
3. **If** m is not acceptable **then Goto** 1.
4. Compute $H_D(x)$.
5. **If** $H_D(x)$ has no roots over GF(q) **then Goto** 1.
6. **Set** $j \leftarrow$ *root of* $H_D(x)$ over GF(q),
7. **Set** $E \leftarrow \left[y^2 = x^3 + 3kx + 2k\right]$ **where** $k = j/(1728 - j)$.
8. **If** $\#E(q) = m$ **then Output** E.
9. **Set** $E \leftarrow \left[y^2 = x^3 + 3kc^2x + 2kc^3\right]$ **where** $k = j/(1728 - j)$ **and** c is any quadratic non-residue over GF(q).
10. **Output** E.

2 Pairings

2.1 Definitions

Let E be an elliptic curve over GF(q).

Definition 4. Let $\mathbf{P}$ be a point of order p where $p \mid \#E(q^\alpha)$. We define such point $\mathbf{P}$ as p-torsion point.

Notation. $E(q^\alpha)[p] \coloneqq \{\mathbf{P} \in E(q^\alpha) \mid \text{order}(\mathbf{P}) = p\}$ - the set of all p-torsion points.

Definition 5. Let p be a large prime divisor of $\#E(q)$. We call an integer k an embedding degree of $E(q)$ if k is a minimal integer s.t. $p \mid q^k - 1$ and $p \nmid q^j - 1$ forall $j \in [1, k-1]$.

Theorem 6. *If E is an elliptic curve over $GF(q)$, and p be a large prime divisor of $\#E(q)$, then $E(q^\alpha)[p] \subset E\left(q^k\right)$ forall α where k is an embedding degree. In other words, all p-torsion points are in $E\left(q^k\right)$.*

Corollary. *Let $\mathbf{P} \in E(q)[p]$, k be an embedding degree. There exists a p-torsion point $\mathbf{Q} \in E\left(q^k\right)[p] \setminus E(q)$. In other words, two linearly independent p-torsion points exist.*

Notation. $\Theta\left(q^k\right)[p] \coloneqq \left\{e \in \mathrm{GF}\left(q^k\right) \mid e^p = 1\right\}$: p-th roots of unity.

Definition 7. We define an asymmetric *Pairing* as a bilinear non-degenerate map as follows.

$$f: E(q)[p] \times E\left(q^k\right)[p] \rightarrow \Theta\left(q^k\right)[p],$$

Bilinear. $f([a]\mathbf{P}, [b]\mathbf{Q}) = (f(\mathbf{P}, \mathbf{Q}))^{ab}$ for any integers a, b.

Non-degenerate. $\begin{cases} [\forall \mathbf{P}, f(\mathbf{P}, \mathbf{Q}) = 1 \iff \mathbf{Q} = \mathcal{O}] \\ [\forall \mathbf{Q}, f(\mathbf{P}, \mathbf{Q}) = 1 \iff \mathbf{P} = \mathcal{O}] \end{cases}$.

Definition 8. A *rational function* of two variables is a function of type $f(x,y)/g(x,y)$ where $f, g \in \mathrm{GF}(q^\alpha)[x, y]$ polynomials accepting two variables.

Definition 9. *Divisors of rational functions.* We wish to record zeros and poles $\mathbf{P}_1, ..., \mathbf{P}_n$ of orders $a_1, ..., a_n$ (where $\mathbf{P}_i$ is a zero of order a_i if $a_i > 0$, and is a pole of order $-a_i$ otherwise), then one can define:

$$D = a_1 \langle \mathbf{P}_1 \rangle + \cdots + a_n \langle \mathbf{P}_n \rangle.$$

[10], [16]

2.2 The Weil Pairing

We define the Weil pairing f as follows. For $\mathbf{P} \in E(q)[p]$ and $\mathbf{Q} \in E\left(q^k\right)[p]$, choose any $\mathbf{R}, \mathbf{S} \in E\left(q^k\right)[p]$ s.t. $\mathbf{S} \notin \{\mathbf{R}, \mathbf{P}+\mathbf{R}, \mathbf{P}+\mathbf{R}-\mathbf{Q}, \mathbf{R}-\mathbf{Q}, \mathcal{O}\}$. Let $f_{\mathbf{P}}$ be a rational function with divisor $p\langle \mathbf{P}+\mathbf{R}\rangle - p\langle \mathbf{R}\rangle$, and $f_{\mathbf{Q}}$ be a rational function with divisor $p\langle \mathbf{Q}+\mathbf{S}\rangle - p\langle \mathbf{S}\rangle$. Define,

$$f(\mathbf{P}, \mathbf{Q}) = \frac{f_{\mathbf{P}}(\mathbf{Q}+\mathbf{S})/f_{\mathbf{P}}(\mathbf{S})}{f_{\mathbf{Q}}(\mathbf{P}+\mathbf{R})/f_{\mathbf{Q}}(\mathbf{R})}.$$

[10], [16]

The standard way of calculating Weil's pairing is by implementing Miller's algorithm which calculates $f_{\mathbf{P}}(\mathbf{Q})$ [17].

Unfortunately our attempts at implementing this pairing strategy have yielded inferior performance results, thence we focus next on the Tate pairing.

2.3 The Tate Pairing

We define the Tate pairing f as follows. For $\mathbf{P} \in E(q)[p]$ and $\mathbf{Q} \in E\left(q^k\right)[p]$, choose any point $\mathbf{R} \in E\left(q^k\right)[p]$ s.t. $\mathbf{R} \notin \{\mathbf{P}, \mathbf{P}-\mathbf{Q}, -\mathbf{Q}, \mathcal{O}\}$. Let $f_{\mathbf{P}}$ be a rational function with divisor $p\langle\mathbf{P}\rangle$. Delineate,

$$f(\mathbf{P}, \mathbf{Q}) = f_{\mathbf{P}}(\mathbf{Q}+\mathbf{R}) / f_{\mathbf{P}}(\mathbf{R}).$$

Remark. In order to get correct values of Tate pairing, one must perform exponentiation of the final result by $\left(q^k-1\right)/p$. This can take a substantial amount of time, but we heavily optimized this process, as explained in the next sub-section.

Notation.

1. $L_{\mathbf{A},\mathbf{B}}(\mathbf{C})$ is the result of the substitution of $\mathbf{C}$ into equation of the straight line which passes through $\mathbf{A}, \mathbf{B}$.
2. $T_{\mathbf{A}}(\mathbf{C})$ is the result of the substitution of $\mathbf{C}$ into a tangent line at $\mathbf{A}$.
3. $V_{\mathbf{A}}(\mathbf{C})$ is the result of the substitution of $\mathbf{C}$ into a vertical line through $\mathbf{A}$.

Algorithm 7 Naive implementation of Miller's algorithm for the Tate pairing.

Input: Appropriate *Points* $\mathbf{P}, \mathbf{Q}$.
Output. $x = f_{\mathbf{P}}(\mathbf{Q})$.

1. **Let** $b_k b_{k-1} \cdots b_1 b_0$ **be the binary representation of** p**,**
2. **Set** $x \leftarrow 1$**,**
3. **Set** $\mathbf{Z} \leftarrow \mathbf{P}$**.**
4. **For** $i = k-1$ **downto** 0 **by** 1 **do**
   - 4.1 **Set** $x \leftarrow x^2 \cdot T_{\mathbf{Z}}(\mathbf{Q}) / V_{2\mathbf{Z}}(\mathbf{Q})$**,**
   - 4.2 **Set** $\mathbf{Z} \leftarrow 2\mathbf{Z}$**.**
   - 4.3 **If** $b_i = 1$ **then**
     - i. **Set** $x \leftarrow x \cdot L_{\mathbf{Z},\mathbf{P}}(\mathbf{Q}) / V_{\mathbf{Z}+\mathbf{P}}(\mathbf{Q})$**,**
     - ii. **Set** $\mathbf{Z} \leftarrow \mathbf{Z}+\mathbf{P}$**.**
5. **Output** x**.**

[17]

In the next sub-section we gain a great performance boost due to optimizations of Miller's algorithm and usage of Elliptic Nets.

2.4 Optimizations

Optimization 1. The naive algorithm performs divisions in every iteration. We would like to accumulate denominators in an auxiliary variable so that while mathematically performing the same operation using multiplications, we issue a division once when the algorithm ends. This optimization applies to the Weil Pairing as well.

Algorithm 8 Slightly optimized implementation of Miller's algorithm for the Tate pairing.

Input: Appropriate *Points* $\mathbf{P}, \mathbf{Q}$.
Output. $x = f_{\mathbf{P}}(\mathbf{Q})$.

1. **Let** $b_k b_{k-1} \cdots b_1 b_0$ **be the binary representation of** p,
2. **Set** $x \leftarrow 1$,
3. **Set** $d \leftarrow 1$,
4. **Set** $\mathbf{Z} \leftarrow \mathbf{P}$.
5. **For** $i = k-1$ **downto** 0 **by** 1 **do**
   - 5.1 **Set** $x \leftarrow x^2 \cdot T_{\mathbf{Z}}(\mathbf{Q})$,
   - 5.2 **Set** $d \leftarrow d \cdot V_{2\mathbf{Z}}(\mathbf{Q})$,
   - 5.3 **Set** $\mathbf{Z} \leftarrow 2\mathbf{Z}$.
   - 5.4 **If** $b_i = 1$ **then**
     - i. **Set** $x \leftarrow x \cdot L_{\mathbf{Z},\mathbf{P}}(\mathbf{Q})$,
     - ii. **Set** $d \leftarrow d \cdot V_{\mathbf{Z}+\mathbf{P}}(\mathbf{Q})$,
     - iii. **Set** $\mathbf{Z} \leftarrow \mathbf{Z} + \mathbf{P}$.
   - 5.5 **Set** $d \leftarrow d^2$.
6. **Output** x/d.

Optimization 2. The functions L, T, V used to work with points in affine coordinates with the formulae below for $\mathbf{A} = (x_0, y_0)$, $\mathbf{B} = (x_1, y_1)$ and $\mathbf{C} = (u, v)$.

$$\begin{aligned} L_{\mathbf{A},\mathbf{B}}(\mathbf{C}) &= (x_0 - x_1)\, v - (y_0 - y_1)\, u - (x_0 y_1 - x_1 y_0), \\ T_{\mathbf{A}}(\mathbf{C}) &= \left(3x_0^2 + a\right)(u - x_0) - 2y_0\,(v - y_0), \qquad [1] \\ V_{\mathbf{A}}(\mathbf{C}) &= u - x_0. \end{aligned}$$

The naive algorithm must transform points into ones in affine coordinates, which is costly due to division routines. We show next the bare essence of the optimization by allowing mixed coordinates calculation as follows. We assume $\mathbf{A} = (x_0, y_0, z_0)$ in Jacobian projective coordinates and $\mathbf{B}$, $\mathbf{C}$ are the same, and thus achieve:

$$\begin{aligned} L_{\mathbf{A},\mathbf{B}}(\mathbf{C}) &= \left(z_0^4 x_0 - z_0^6 x_1\right) v - \left(z_0^3 y_0 - z_0^6 y_1\right) u - \left(z_0^4 x_0 y_1 - z_0^3 x_1 y_0\right), \\ T_{\mathbf{A}}(\mathbf{C}) &= \left(3x_0^2 + z_0^4 a\right)\left(z_0^2 u - x_0\right) - 2y_0\left(z_0^3 v - y_0\right), \\ V_{\mathbf{A}}(\mathbf{C}) &= z_0^4 u - x_0. \end{aligned}$$

This conserves such divisions by performing additional multiplications instead. Clearly this optimization applies to Weil or Tate pairing also, when using Miller's algorithm.

Optimization 3 - Elliptic Nets. We would like to replace Miller's algorithm with a better approach, for the Tate pairing, with Shipsey-Stange algorithm which utilizes elliptic nets. For more mathematical background about elliptic nets, consult [6], [9].

Let $E : y^2 = x^3 + ax + b$ be an elliptic curve over a field GF (q^α). Let $\mathbf{P} = (x_0, y_0) \in E(q^\alpha)[p]$, $\mathbf{Q} = (x_1, y_1) \in E(q^\alpha)$. Let us define these constants: $A = (x_0 - x_1)^{-1}$, $B = \left((2x_0 + x_1)(x_0 - x_1)^2 - (y_0 + y_1)^2\right)^{-1}$, $C = (2y_0)^{-1}$.

We now define the sequence c_k by $c_{-2} = -2y_0$, $c_{-1} = -1$, $c_0 = 0$, $c_1 = 1$, $c_2 = 2y_0$,

$$\begin{array}{lcl} c_3 & = & 3x_0^4 + 6ax_0^2 + 12bx_0 - a^2, \\ c_4 & = & 4y_0\left(x_0^6 + 5ax_0^4 + 20bx_0^3 - 5a^2x_0^2 - 4abx_0 - 8b^2 - a^3\right), \\ c_{2k-1} & = & c_{k+1}c_{k-1}^3 - c_{k-2}c_k^3, \\ c_{2k} & = & C\left(c_k c_{k+2}c_{k-1}^2 - c_k c_{k-2}c_{k+1}^2\right). \end{array}$$

and the sequence d_k by $d_0 = 1$, $d_1 = 1$, $d_2 = (2x_0 + x_1) - \left(\frac{y_1 - y_0}{x_1 - x_0}\right)^2$,

$$\begin{array}{lcl} d_{2k-1} & = & d_{k+1}d_{k-1}c_{k-1}^2 - d_k^2 c_{k-2}c_k, \\ d_{2k} & = & d_{k+1}d_{k-1}c_k^2 - d_k^2 c_{k-1}c_{k+1}, \\ d_{2k+1} & = & A\left(d_{k+1}d_{k-1}c_{k+1}^2 - d_k^2 c_k c_{k+2}\right), \\ d_{2k+2} & = & B\left(d_{k+1}d_{k-1}c_{k+2}^2 - d_k^2 c_{k+1}c_{k+3}\right). \end{array}$$

The *Simplified Tate Pairing*, which we use for cryptographic properties, is defined for even α, and

$$f(\mathbf{P}, \mathbf{Q}) = f_{\mathbf{P}}(\mathbf{Q}) = d_p.$$

Algorithm 9 Simplified Tate pairing by elliptic nets

Input: Appropriate *Points* $\mathbf{P}, \mathbf{Q}$.
Output. $x = f(\mathbf{P}, \mathbf{Q})$.

1. **Let** $b_t b_{t-1} \cdots b_1 b_0$ **be the binary representation of** p,
2. **Let** $k \leftarrow 1$.
3. **Compute** A, B, C **and** *sequences* $(c_{-2}, ..., c_5$ **and** $d_0, ..., d_2)$.
4. **For** $j = t - 1$ **downto 0 by 1 do**
   - 4.1 **Compute** *sequences* $(c_{2k-3+b_j}, ..., c_{2k+4+b_j}$ **and** $d_{2k-1+b_j}, ..., d_{2k+1+b_j})$.
   - 4.2 **Let** $k \leftarrow 2k + b_j$.
5. **Output** d_p.

[17]

Optimization 3.1. We can reduce the amount of shared expressions throughout the algorithm, hence improving efficiency by reducing subsequent calculations via better revising the earlier definitions: $S_k = c_k^2$, $T_k = c_{k-1}c_{k+1}$, $U_k = d_k^2$, $V_k = d_{k-1}d_{k+1}$,

$$\begin{array}{lcl} c_{2k-1} & = & T_k S_{k-1} - T_{k-1}S_k, \\ c_{2k} & = & C\left(T_{k+1}S_{k-1} - T_{k-1}S_{k+1}\right), \end{array} \qquad \begin{array}{lcl} d_{2k-1} & = & V_k S_{k-1} - U_k T_{k-1}, \\ d_{2k} & = & V_k S_k - U_k T_k, \\ d_{2k+1} & = & A\left(V_k S_{k+1} - U_k T_{k+1}\right), \\ d_{2k+2} & = & B\left(V_k S_{k+2} - U_k T_{k+2}\right). \end{array}$$

[17]

Optimization 3.2. Assuming $\mathbf{P} \in E(q)[p]$, we can take optimization 3 one step further: for all k, $[c_k \in \mathrm{GF}(q)]$ holds - therefore effectively positively affecting every computation involving S and T in the much smaller field.

Optimization 4. A final exponentiation must be conducted in every algorithm for Tate pairing, which is done by computing $(f(\mathbf{P},\mathbf{Q}))^{(q^\alpha-1)/p}$. A naive approach, using default a exponentiation method, would yield a slower calculation time, caused by the large number of bits of the exponent. The trick we exploit is held by using a Frobenius map, stepwise exponentiating the result by cyclotomic divisors of $q^\alpha - 1$; and finishing by performing a default exponentiation by a much smaller exponent. This optimization is valid for every algorithm for computing Tate pairing.

Example for $\alpha = 6$. Let $a = f(\mathbf{P},\mathbf{Q})$. We wish to compute $a^{(q^6-1)/p}$, our first step is to compute $b = a^{(q^4+q^3-q-1)} = \frac{\left(u_0+u_1x^{q^4}+\cdots+u_5x^{5q^4}\right)\left(u_0+u_1x^{q^3}+\cdots+u_5x^{5q^3}\right)}{(u_0+u_1x^q+\cdots+u_5x^{5q})a}$ where $a = u_0 + u_1x + \cdots + u_5x^5$. Note that every appropriate power of x^q may be precomputed. Then we use a default exponentiation method to calculate $b^{(q^2-q+1)/p}$.

2.5 Pairing-friendly curves

2.5.1 MNT curves

One of the families of curves we implemented in our library which is described in [11] - MNT curves, which we can construct using the CM method.

Our desire is to utilize Algorithm 6, alas we require special triples for the first step of the algorithm, and so we attain an equation of the kind $x^2 - gy^2 = f$ - the generalized Pell equation, which can be solved using specialized techniques [12]. An acceptable $\#E(q)$ has a large prime factor or prime, ideally. The fully formal algorithm is given in [11].

2.5.2 Barreto and Naehrig curves

The second family of curves we employ are discussed in [17] as "Type F" curves. Define the following polynomials:

$$\begin{aligned} q(x) &= 36x^4 + 36x^3 + 24x^2 + 6x + 1, \\ t(x) &= 6x^2 + 1. \end{aligned}$$

We acquire the following algorithm for generating such curves:

Algorithm 10 Generate a Barreto and Naehrig curve.

`No Input.`
`Output.` A curve E over $\mathrm{GF}(q)$ with prime $\#E(q)$.

1. **Set** $x \leftarrow$ **random** *integer*,
2. **Set** $q \leftarrow q(x)$.
3. **If** q is not prime **then Goto** 1.
4. **Set** $m \leftarrow q - t(x) + 1$.
5. **If** m is not prime **then Goto** 1.
6. **Set** $k \leftarrow$ **random** *element in* $\mathrm{GF}(q)$,
7. **Set** $E \leftarrow \left[y^3 = x^3 + k\right]$.
8. **If** $\#E(q) \neq m$ **then Goto** 6.
9. **Output** E.

3 Cryptographic use

3.1 Diffie-Hellman groups

Let G_1, G_2 be groups.

Definition 10. Computational co-Diffie-Hellman problem (co-CDH). Input: $g_2, g_2^a \in G_2$ and $h \in G_1$, Output: $h^a \in G_1$.

Definition 11. Decision co-Diffie-Hellman problem (co-DDH). Input: $g_2, g_2^a \in G_2$ and $h, h^b \in G_1$, Output: $a \stackrel{?}{=} b$; if $a = b$ then (g_2, g_2^a, h, h^a) is said to be a co-Diffie-Hellman tuple.

Note. When $G_1 = G_2$, these problems are reduced to the standard Diffie-Hellman problems.

Definition 12. Gap co-Diffie-Hellman group pair is a tuple (G_1, G_2) satisfying:
1. Group operations are done in polynomial time.
2. The map $\psi : G_2 \to G_1$ exists and can be computed in polynomial time.
3. co-DDH on (G_1, G_2) can also be solved in polynomial time.
4. co-CDH on (G_1, G_2) is hard.

Theorem. *In our case of elliptic curves, let* $G_1 = \langle \mathbf{P} \rangle$ *and* $G_2 = \langle \mathbf{Q} \rangle$ *where* $\mathbf{P} \in E(q)[p]$ *and* $\mathbf{Q} \in E(q^\alpha)[p] \setminus E(q)$*; thus we get the gap co-Diffie-Hellman pair* (G_1, G_2).

[13]

3.2 Short Signature algorithms

Let $G_1 = \langle \mathbf{P} \rangle$ and $G_2 = \langle \mathbf{Q} \rangle$ as described before, and let e be the pairing map.

3.2.1 BLS scheme [13]

We use a hash function, $MapToGroup: \{0,1\}^* \to G_1$.

Algorithm 11 BLS Key Generation

No Input.
Output. PRIVATE KEY $x \in \mathbb{Z}_p$, PUBLIC KEY $\mathbf{V} \in G_2$.

1. **Set** $x \leftarrow$ **random** *integer* **in** $\mathbb{Z}_p$,
2. **Set** $\mathbf{V} \leftarrow x\mathbf{Q}$.
3. **Output** $(x, \mathbf{V})$.

Algorithm 12 BLS Signing

Input. PRIVATE KEY $x \in \mathbb{Z}_p$, MESSAGE $M \in \{0,1\}^*$.
Output. SIGNATURE $s \in \mathrm{GF}(q)$.

1. **Set** $\mathbf{R} \leftarrow MapToGroup(M)$,
2. **Set** $\sigma \leftarrow x\mathbf{R}$,
3. **Output** *x-coordinate* **of** σ.

Algorithm 13 BLS Verification

Input. PUBLIC KEY $\mathbf{V} \in G_2$, MESSAGE $M \in \{0,1\}^*$, SIGNATURE $s \in \mathrm{GF}(q)$.
Output. SIGNATURE VALIDITY.

1. **Try (Compute** y **in** $\mathrm{GF}(q)$ **such that** (s, y) **in** $E(q)$**)**
   If Invalid (Output *INVALID***).**
2. **Set** $\sigma \leftarrow (s, y)$.
3. **If (***order* **of** σ**)** $\neq p$ **then**
   3.1 **Output** *INVALID*.
4. **Set** $\mathbf{R} \leftarrow MapToGroup(M)$.
5. **If** $e(\sigma, \mathbf{Q}) = e(\mathbf{R}, \mathbf{V})$ **or** $(e(\sigma, \mathbf{Q}))^{-1} = e(\mathbf{R}, \mathbf{V})$ **then**
   5.1 **Output** *VALID*.
6. **Output** *INVALID*.

3.2.2 Modified ZSS scheme [14]

The key generation routine is unchanged. Let H be a hashing function.

Algorithm 14 Modified ZSS Signing

Input. PRIVATE KEY $x \in \mathbb{Z}_p$, MESSAGE $M \in \{0,1\}^*$.
Output. SIGNATURE $s \in \mathrm{GF}(q)$.

1. **Set** $\mathbf{S} \leftarrow \frac{1}{H(m)+x}\mathbf{P}$.
2. **Output** *x-coordinate* **of** $\mathbf{S}$.

Algorithm 15 Modified ZSS Verification

Input. PUBLIC KEY $\mathbf{V} \in G_2$, MESSAGE $M \in \{0,1\}^*$, SIGNATURE $s \in \mathrm{GF}(q)$.
Output. SIGNATURE VALIDITY.

1. **Try (Compute** y **in** $\mathrm{GF}(q)$ **such that** (s,y) **in** $E(q)$**)**
   If Invalid (Output *INVALID***).**
2. **Set** $\mathbf{S} \leftarrow (s,y)$.
3. **If (***order* **of** $\mathbf{S}$**)** $\neq p$ **then**
   3.1 **Output** *INVALID.*
4. **Set precomputed** $\Omega \leftarrow \left\{e(\mathbf{P},\mathbf{Q}), (e(\mathbf{P},\mathbf{Q}))^{-1}\right\}$.
5. **If** $e(\mathbf{S}, H(m)\mathbf{Q}+\mathbf{V})$ **in** Ω **then //// Modification is here ////**
   5.1 **Output** *VALID.*
6. **Output** *INVALID.*

3.2.3 Comparison

Scheme	Sign	Verify
BLS	1 *MapToGroup*, 1 Multiplication	1 *MapToGroup*, 2 Pairing Operations
Modified ZSS	1 Inversion, 1 Multiplication	1 Pairing Operation, 1 Multiplication

Remark. Due to many point operation optimizations, pairing operations are relatively more costly, hence Modified ZSS is more efficient in our setting.

3.3 Strengths

For average 160-bit field size q, both types below, in terms of Elliptic Curve Discrete Log, are sufficiently cryptographically strong.

MNT ($\alpha = 6$)**.** After the pairing operation is done, the computation results are in field of average size $160 * 6 = 960$ bits; thence, in terms of the standard Discrete Log, we can achieve the same security as 1024-bit RSA, when raising the field size q to 170 bits for example.

Barreto and Naehrig curves ($\alpha = 12$)**.** Similarly, the average field size after pairing is $160 * 12 = 1920$ bits; hence we get almost the same security level as 2048-bit RSA.

Part III
Implementation

4 Documentation

The project was implemented as a `C++` library, **LibECq**; to be of use for other projects on top of it. We used `GCC 4.9.1`, `NTL 6`, `GNU GMP` and `GNU MPFR` on a *UNIX 03* conforming system.

4.1 Library overview

LibECq.hpp

1. One must issue `#include "LibECq.hpp"` at the top of their program in order for the library classes and definitions to become available to all of their program's elements.
2. Particular library elements and *Settings* can be included only if their appropriate MACRO definition is present and placed *above* the mentioned `#include` line above.

Remark. The order of these MACROs above the inclusion has no significance.

Example.

```
#define NAMESPACE_LIBECQ
#define LIBECQ_SHORT_SIGNATURES
#define UTILITIES_SNIPPETS
#include "LibECq.hpp"
```

4.2 Settings - global library control macros (prior to #include)

In this section, we use the notation *Set* or *Issue* to direct the user to define a macro with a value or define a macro so that it only exists, appropriately.

Randomization.

`LIBECQ_RANDOM_SEED` ***natural.*** Set to determine the random seed.

`LIBECQ_RANDOM_DEFAULT_SEED`, `LIBECQ_NO_RANDOM.` Issue to use a preset seed or disable the mechanism.

Interaction with NTL.

`LIBECQ_DONT_INIT_P`, `LIBECQ_P` ***natural.***

- Issue to prevent the library from initializing `NTL::ZZ_p` with the `LIBECQ_P` value before the `main` procedure is executed.
- Set `LIBECQ_P` to direct the library what initial `NTL::ZZ_p` to use, instead of the default value of 19.

Modulus.

`BITS` *`natural`* , `MIN_BITS` *`natural`*, `MAX_BITS` *`natural`*. If `BITS` is not set then the default set of $(160, 170, 200)$ appropriately will be used, otherwise set `BITS` as well as `MIN_BITS` and `MAX_BITS` to control these sizes used by MNT curves.

Linking library components.

`LIBECQ_EVERYTHING`, `NAMESPACE_ALL`, `UTILITIES_ALL`, `LIBECQ_SHORT_SIGNATURES`, `LIBECQ_PAIRING`, `UTILITIES_COMPLEX`, `UTILITIES_CORNACCHIA`, `UTILITIES_HILBERT`, `UTILITIES_PELL`, `UTILITIES_POLYNOMIAL`, `UTILITIES_SHA3`, `UTILITIES_SNIPPETS`, `UTILITIES_STRINGBYTES`, `LIBECQ_DEBUG`. `LIBECQ_EVERYTHING` issues `UTILITIES_ALL` and `NAMESPACE_ALL` which, respectively, issue the rest of the listed macros, which in turn reveal the issued classes to the user's program - and applies use of the various namespaces such as `std`, `NTL`, `libecq` and `libecq::utilities`.

Prime extraction max divisor setting.

`LIBECQ_ENABLE_MAX_DIVISOR`, `MAX_DIVISOR` *`natural number`*. Issue `LIBECQ_ENABLE_MAX_DIVISOR` to use `MAX_DIVISOR` (set to 1000 by default) as an upper bound on a number k when $n = pk$ for a large prime divisor of n, p; when factoring a number n with `PrimeExtractor::extract`.

Debugging.

`DNDEBUG`, `LIBECQ_DEBUG`, `DEBUG(`*`object stream`*`)`, `ASSERT(`*`expr`*`)`, `STAR()`, `MU()`, `PSI()`, `DELTA()`, `SIGMA()`, `DOUBLE_PIT()`. Issue `DNDEBUG` to disable these debugging macros, or issue `LIBECQ_DEBUG` to link them.

`LIBECQ_DEBUG`, `LIBECQ_INTERNAL_LIBRARY_ADDITIONAL_CHECKS_ENABLED`, `LIBECQ_INTERNAL_LIBRARY_DEBUG_DATA_ENABLED`. One may wish to use these macros to enable more extensive testing which impact time complexity for safety or parameter examining.

4.3 Classes and the inheritance model

4.3.1 Points over finite fields

- `Point, PointEx` for use with `ECq` and `ECqEx` respectively.

4.3.2 Elliptic curves

- `ECq, K12` » `ECqBase (abstract)` used working with EC over GF (q) of the form $y^2 = x^3 + ax + b$ (or $Y^2 = X^3 + aXZ^4 + bZ^6$), or over the extension field with $\alpha = 12$ of the form $y^2 = x^3 + b$ (or $Y^2 = X^3 + bZ^6$), respectively.
- `MNT` » `K12` for working with MNT curves.

4.3.3 Pairing

- `Weil, Tate, TateFast` » `Pairing (⌘MetaPairing)`,
- `WeilMNT, TateMNT, TateFastMNT` » `PairingMNT` » `Pairing` used for assigning a short signature object with a specified pairing scheme.

4.3.4 Short Signatures

- `ShortSignature` used for generating, verifying and testing of short digital signatures. `TateFast` is used as the default pairing scheme.

4.3.5 Time measuring

- `Timer` - a stopwatch used to precisely measure difference in time with very little absolute error in nano seconds (dependents on `NTL::GetTime()` call time).

4.3.6 Utilities

- `PrimeExtractor` - see *Prime extraction max divisor setting* on section 4.2.
- `Complex` - a complex number manipulation class.
- `Cornacchia` - a class which can be used to solve equations of the *Cornacchia* form (assuming p is prime and $-d$ is a quadratic residue mod p) - details in section 1.3.2 equation (1).
- `Pell` - solves *Pell* equations of the form $x^2 - Dy^2 = n$ for $n^2 < d$ where d is square-free. Moreover in [2].
- `Hilbert` - presented in section 1.3.2.
- `Polynomial` - handling and manipulation of polynomials with complex coefficients with control over precision.
- `SHA3` - generates, validates and compares *SHA3* hashes to/from strings represented in Hex; and supports a streaming approach also.
- `Snippets` - `Triple(obj,obj,obj)`, `BEEP()`, `endll`, `m_ary_decompose(ZZ,int)`, `NAF(ZZ)` – a handful of helper functions/classes.
- `StringBytes` - handles accurate conversions from raw *aligned* byte arrays to `C++` strings and vice versa.

4.4 Exceptions

- `ECCreator_Invalid` » `ECC_Exception` » `LibECq_Exception` » `std::exception`
- `ECqEx_Wrong_Field_Size` » `ECqEx_Exception` » `LibECq_Exception` » `std::exception`
- `Pairing_Line_Zero` » `Pairing_Exception` » `std::exception`

4.5 Usage examples

Defining a curve and printing it.

```
MNT ec;
cout << ec << endl;
```

Creating a signature and verifying it.

```
ShortSignature ss(ec,true);
ZZ_p sig = ss.sign("Pi");
bool result = ss.verify_my_public_key("Pi",sig);
```

Remark.

1. In this example, the `ShortSignature` class constructor takes an optional boolean parameter, if true, the class generates a private and public key pair and sets them within the `ShortSignature` object.

2. Also, the issued verification function in this example uses the user's generated public key. If one needs to verify a specified public key, they can do so by issuing `ss.verify("Pi",sig,public_key)` for example.

Using more than one elliptic curve.

```
MNT ec1;
ECqEx ec2;
ShortSignature ss1(ec1,true);
ShortSignature ss2(ec2,true);
*ec1;
ZZ_p sig1 = ss1.sign("Pi");
bool result1 = ss1.verify_my_public_key("Pi",sig1);
ec2.use();
ZZ_p sig2 = ss2.sign("Rain");
bool result2 = ss2.verify_my_public_key("Rain",sig2);
```

Important Remark. Due to limitations of the `NTL` library, one must switch to the appropriate elliptic curve before utilizing it if using more than one in their program, by either issuing `*ec` or `ec.use()`. Anywhere `ec` can be placed, so can its switched call.

Using a specified pairing rather than the default TateFast or TateFastMNT.

```
MNT ec1;
ECqEx ec2;
ShortSignature ss1(WeilMNT(*ec1));
ShortSignature ss2(Tate(*ec2),true);
```

5 Performance

We measured average times on a 2.8 GHz quad core *UNIX-03* conforming system.

5.1 Elliptic curve arithmetic

	Addition	Doubling	Scalar Mult	.. (fixed point)	Random Point
Base Field (160 *bits*)	$4.110ns$	$3.447ns$	$0.783ms$	–	$37.008ns$
MNT Extended (1020 *bits*)	$46.209ns$	$40.269ns$	$11.221ms$	$3.732ms$	$49.428ms$
Barreto and Naehrig (1920 *bits*)	$145.739ns$	$129.832ns$	$33.175ms$	$11.417ms$	$0.332s$

Addition » Average time for a single point addition operation

Doubling » Average time for a single point doubling operation

Scalar Multiplication » Average time for a point Scalar Multiplication operation

Scalar Multiplication (fixed point) » Same as above but using precomputed data

Random Point » Average time for finding a random point on a given curve

5.2 Pairing and Short Signatures

	Initialization	Tate-Miller	**Tate-Nets**	Key Gen	Sign	Verify
MNT (170 *bits*)	$0.246s$	$59.939ms$	**6.963ms**	$3.956ms$	$0.539ms$	$11.126ms$
Barreto and Naehrig (160 *bits*)	$0.799s$	$333.708ms$	**29.578ms**	$11.714ms$	$0.488ms$	$41.932ms$

Initialization » Average time for creating a curve with all needed parameters

Tate-Miller » Average time for a single Tate pairing operation using Miller's algorithm

Tate-Nets » Average time for a single Tate pairing operation using elliptic nets

Key Generation, Sign, Verify » Average time for a Short Signature operation

Remark. Some of our results are better than some presented in other articles, such as [6], where in that example their execution time for computing Tate pairing via elliptic nets takes about $130ms$.

Part IV
Conclusions

We researched methods and approaches for generating and verifying signatures of twice shorter than the standard signature length, while implementing many optimizations which origin from mathematical background such as theory of finite fields, algorithm understanding by modifications applied to them, numerical analysis for understanding the propagation of errors and so on.

Algorithm modifications.

Miller algorithm. We applied Jacobian projective coordinates to achieve mixed Line calculations, needed in Miller's algorithm along with denominator accumulating technique for reducing division operations notably.

ZSS Short Signature Algorithm. We extended the base algorithm's underlying group usage with two different groups, which comply to the Gap Diffie-Hellman group pair; along with pre-computation of various constant pairing values for enhanced speedups.

Final exponentiation of Tate pairing. We altered the strategy of the final exponentiation by adding an additional step in lieu of using a default exponentiation in the final stage.

We gained useful experience and knowledge of real-world algorithms and topics as well as interesting abstract mathematical concepts, which contribute to the understanding of our implementation thus allowing us to tweak and enhance it significantly.

Part V
Bibliography

1. IEEE - ***IEEE P1363 / D13. Standard Specifications for Public Key Cryptography, Annex A, Number-Theoretic Background*** *(1999).*

2. I.F. Blake, G. Seroussi, N.P. Smart - ***Smart Elliptic Curves in Cryptography*** *(1999).*

3. Richard Crandall, Carl Pomerance - ***Prime Numbers A Computational Perspective Second Edition*** *(2005).*

4. Andrew V. Sutherland - ***Computing Hilbert class polynomials with the CRT method*** *(2008).*

5. Sedat Akleylek, Barış Bülent Kirlar, Ömer Sever, Zaliha Yüce - ***Short Signature Scheme From Bilinear Pairings*** *(2011).*

6. Naoki Ogura, Naoki Kanayama, Shigenori Uchiyama, Eiji Okamoto - ***Cryptographic Pairings Based on Elliptic Nets*** *(2010).*

7. Kenneth Barr and Krste Asanović - ***Energy Aware Lossless Data Compression*** *(2003).*

8. Elisabeth Oswald - ***Introduction to Elliptic Curve Cryptography*** *(2005).*

9. Katherine E. Stange - ***The Tate Pairing via Elliptic Nets*** *(2007).*

10. Victor S. Miller - ***The Weil Pairing, and Its Efficient Calculation*** *(2004).*

11. Michael Scott, Paulo S. L. M. Barreto - ***Generating more MNT elliptic curves*** *(2004).*

12. John P. Robertson - ***Solving the generalized Pell equation*** $x^2 - Dy^2 = N$ *(2004).*

13. Dan Boneh, Ben Lynn, Hovav Shacham - ***Short Signatures from the Weil Pairing*** *(2001).*

14. Fangguo Zhang, Reihaneh Safavi-Naini, Willy Susilo - ***An Efficient Signature Scheme from Bilinear Pairings and Its Applications*** *(2004).*

15. Gora Adj, Francisco Rodríguez-Henríquez - ***Square root computation over even extension fields*** *(2012).*

16. Alex Edward Aftuck - ***The Weil Pairing on Elliptic Curves and Its Cryptographic Applications*** *(2011).*

17. Ben Lynn - ***ON THE IMPLEMENTATION OF PAIRING-BASED CRYPTOSYSTEMS*** *(2007).*

18. Darrel Hankerson, Alfred Menezes, Scott Vanstone - ***Guide to Elliptic Curve Cryptography*** *(2003).*

19. Cheol Min Park, Myung Hwan Kim, Moti Yung - ***A Remark on Implementing the Weil Pairing*** *(2005).*

20. J.H. Silverman - ***Advanced Topics in the Arithmetic of Elliptic Curves*** *(1994).*