Batch Verification of Elliptic Curve Digital Signatures

by

Michael Wesolowski

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics & Optimization

Waterloo, Ontario, Canada, 2015

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis investigates the efficiency of batching the verification of elliptic curve signatures. The first signature scheme considered is a modification of ECDSA proposed by Antipa et al. along with a batch verification algorithm by Cheon and Yi. Next, Bernstein's EdDSA signature scheme and the Bos-Coster multi-exponentiation algorithm are presented and the asymptotic runtime is examined. Following background on bilinear pairings, the Camenisch-Hohenberger-Pedersen (CHP) pairing-based signature scheme is presented in the Type 3 setting, along with the derivative BN-IBV due to Zhang, Lu, Lin, Ho and Shen. We proceed to count field operations for each signature scheme and an exact analysis of the results is given. When considered in the context of batch verification, we find that the Cheon-Yi and Bos-Coster methods have similar costs in practice (assuming the same curve model). We also find that when batch verifying signatures, CHP is only 11% slower than EdDSA with Bos-Coster, a significant improvement over the gap in single verification cost between the two schemes.

Acknowledgements

I would like to thank my advisor Alfred Menezes for his patience and help in the completion of this thesis. Without his guidance it would not have been possible, and his many revisions improved the quality of this work immeasurably. Thank you to my reading committee members David Jao and Douglas Stinson for their comments and suggestions for improving this work. Thank you to my parents, especially my mother Theresa who has supported me in numerous ways throughout my academic career. Thank you God, for the strength to persevere.

Dedication

This thesis is dedicated to my family. Their constant love and encouragement made my good days great and my bad days bearable. Thanks also to the community at Resurrection College, who showed me friendship and were a joy to be around.

Table of Contents

List of Tables

Chapter 1

Introduction

Digital signatures form an integral part of secure communications. The idea of digital signatures arising from public key cryptography emerged from the paper "New Directions in Cryptography" by Diffie and Hellman in 1976 [49]. The first concrete example of a digital signature scheme came from the RSA encryption scheme [100]. This served as a proof of concept, but did not provide security in practice: Boneh, Joux and Nguyen gave an attack with $\approx 18\%$ success probability that recovers the session key from a basic key establishment protocol that uses textbook RSA [30]. For a survey of attacks on RSA, see [28]. Since their introduction, the repertoire of digital signatures has expanded rapidly, along with mathematical advances in lattices, pairings, and elliptic curves. Although efforts are being made to design cryptographic systems that remain secure with the invention of a large-scale quantum computer [23], elliptic curve schemes remain the most efficient and secure option for many security goals. Elliptic curve topics will command much of the later discussion; for now, we introduce some general notions that we will use throughout the rest of this thesis.

1.1 Digital Signatures

A digital signature scheme is a five-tuple $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ where the following conditions are satisfied [114]:

1. $\mathcal{P}$ is a finite set of possible messages
2. $\mathcal{A}$ is a finite set of possible signatures

3. $\mathcal{K}$, the keyspace, is a finite set of possible keys

4. For each $K \in \mathcal{K}$, there is a signing algorithm $\text{sig}_K \in \mathcal{S}$ and a corresponding verification algorithm $\text{ver}_K \in \mathcal{V}$. The functions $\text{sig}_K, \text{ver}_K : \mathcal{P} \times \mathcal{A} \to \{true, false\}$ satisfy the following for every message $x \in \mathcal{P}$ and for every signature $y \in \mathcal{A}$:

$$\text{ver}_K(x, y) = \begin{cases} true & \text{if } y = \text{sig}_K(x) \\ false & \text{if } y \neq \text{sig}_K(x). \end{cases}$$

Given such a collection, we can sign messages and verify signatures on messages. However, we should also be concerned with the security goals of a signature scheme so that its strength can be evaluated. The attack model adopted in mathematical cryptography allows an adversary (thought of as a computer program) some information, with the goal being to prevent a forgery of some description. The common attacker powers are enumerated in [114] along with the common security goals. We give only the weakest attacker goal paired with the strongest information available.

Definition 1.1 (Chosen-Message Attack)**.** A forger $\mathcal{F}$ is allowed to obtain valid signatures for messages $x_1, x_2, ..., x_N$ of its choosing. That is, $\mathcal{F}$ is given $y_i = \text{sig}_K(x_i)$ for all i.

Definition 1.2 (Existential Unforgeability)**.** A signature scheme is *existentially unforgeable* under chosen-message attack if an adversary is unable to produce a single valid message-signature pair (x, y) where $x \notin \{x_1, \ldots, x_N\}$, when allowed to launch a chosen message attack.

The forger $\mathcal{F}$ must produce a signature on a message that it did not query the function sig_K with. In addition, one usually allows the forger the power to select its messages *adaptively*, so he may choose x_2 after receiving signature y_1 on x_1, as y_1 perhaps gives some information that may be of use to the attacker.

1.2 Batch Verification

Bellare et al. introduced a generic definition of batch verification [18], although practical examples of such schemes appeared earlier [53]. The general idea is to amortize the cost of exponentiation over a large number of instances. That is, given a cyclic group G with generator g of prime order, we have to check that $g^{x_i} = y_i$ for all pairs $(x_i, y_i), i = 1, ..., N$. A correct but inefficient way to do this is to perform n independent exponentiations to the base g. Batch verification of exponentiations is accomplished by computing a single

exponentiation in the group, and only accepts invalid pairs (x_i, y_i) with probability at most $2^{-\ell}$, where ℓ is a security parameter.

Definition 1.3 (Batch Verifier [18])**.** A batch verifier for a relation R is a probabilistic algorithm V that takes as input a batch instance $X = (inst_1, \ldots, inst_N)$ for R, and a security parameter ℓ (independent of N). It satisfies:
1. If X is correct then V outputs 1;
2. If X is incorrect then the probability that V outputs 1 is at most $2^{-\ell}$.

All of the batch verification algorithms considered in this thesis satisfy Definition 1.3. For EdDSA, we verify an equation of the form $S = \sum_{i=0}^{N-1} z_i P_i$, where scalars $z_i \in \{0,1\}^\ell$, $P_i \in \langle g \rangle$, and $\langle g \rangle$ is written additively. Each signature is valid if $P_i = \mathcal{O}$, so we detect forgeries if the sum does not evaluate to $\mathcal{O}$. What is the probability of a forgery not being detected in this batch? That is, what is the probability that some $P_j \neq \mathcal{O}$, but $S = \mathcal{O}$? Since we are working in a cyclic group, if $P_j \neq \mathcal{O}$ then the choice of scalars z_j is determined by the other z_i's. Since the verifier chooses these scalars at random, an attacker has success probability at most $2^{-\ell}$, where ℓ is determined by the size of the sampling set for the scalars.

1.3 Batch Forgery Identification

Batch forgery identification is another service that may be provided with batch verification. At a high level, this includes any method that can identify a forged signature. A trivial method for forgery identification is single verification: one can use the standard verification equation to test every message-signature pair for validity, and determine with certainty that no forgeries are present. If we have many such signatures to verify (as in the general problem), this is not the best we can do. A survey of forgery identification methods was given by Stinson and Zaverucha [119]. In the generic group setting, an algorithm that attempts to find a defective element is called a *group testing algorithm*. Theorem 2.1 of [119] (attributed to Du and Hwang) states that any group testing algorithm is strictly less efficient than single verification if the number of invalid signatures in a batch is greater than or equal to $\approx 38\%$ of the size of the batch, and it is possible to do better when the number of invalid signatures is less than 1/3 of the size of the batch. Bernstein et al. [24] have analyzed existing algorithms and proposed some improved algorithms for forgery identification. The basic idea is to re-use some of the work done when attempting to identify forgeries. For example, if we compute the entire multi-exponentiation and find the result not equal to $\mathcal{O}$, split the batch into two and compute the left multiexponentiation, and the

right half can be computed for free with a single subtraction. This identifies which half of the batch contains the forgery (if there is a single one), and continues recursively down the tree structure until the faulty signature is identified. For more detail and discussion of various algorithmic strategies, see [24].

1.4 Applications

Signature verification is known to be extremely fast in RSA, but the key size required for the 128-bit security level is 3072 bits [7]. Since the best known attacks against the discrete logarithm problem on elliptic curve groups are generic (Pollard Rho), this makes the much smaller key and certificate sizes of ECC an attractive option.

If the cost of verification is taken over the size of the batch, we get some interesting results. In 2011, Rangasamy, Stebila, Boyd and Nieto [97] implemented an anti-Denial of Service (DoS) key exchange scheme with a mix of weak client-based authentication and strong authentication with digital signatures. Since DoS attacks attempt to "consume a server's limited resources such as CPU cycles, memory and network bandwidth" [97], efficiency is a top priority for system designers, and the effort in limiting server computation (to the extent of saving a few modular operations) supports this view. Bernstein notes that such verification-intensive applications can potentially use ECC thanks to batch verification techniques. The advantages of ECC are magnified significantly at higher security levels (shorter keys, faster signing), and thus this option appears to be increasingly popular in practice. For example, Apple uses elliptic curve Diffie-Hellman key exchange over Curve25519 in their operating system [2], among other adopters [6].

Vehicle-to-vehicle (V2V) applications are another example of how ECC is used to provide fast and secure authentication. In a vehicular ad-hoc network (VANET), vehicles communicate with surrounding roadside infrastructure using a wireless communication protocol [98]. Recent efforts to standardize strong authentication in VANET networks with ECC have been encoded within the DSRC standard [8].

Discrete-log based protocols in general benefit from the idea of batching. Bellare et al. give an n-party signature protocol that uses the small exponents test. This scheme is used in teleconferencing, where all participants are connected to a central facility called a *bridge*, which receives signals from the participants, operates on these signals in an appropriate way, and then broadcasts the result back to the participants [18]. Goldberg and Henry [66] note several applications of batch discrete-log zero knowledge proofs (which use batch verification as a basic operation) including symmetric private information retrieval, cryp-

tographic voting systems, and anonymous blacklisting. Thus, fast multi-exponentiation in groups is required in several applications.

1.5 Thesis Outline

The remainder of the thesis is organized as follows. Chapter 2 focuses on the ECDSA* signature scheme and a batch verification technique due to Cheon and Yi. It concludes with a recent practical proposal of Möller and Rupp for caching data to accelerate ECDSA signature verification. Chapter 3 introduces Edwards curves and the EdDSA signature scheme. It begins with a proof of security for the Schnorr signature scheme, which is the basis for EdDSA. The chapter concludes with an examination of batch verification techniques used in EdDSA. Chapter 4 considers the family of Barreto-Naehrig (BN) elliptic curves and the optimal Ate pairing. We apply these curves in the Type 3 variant of the identity-based batch verification scheme due to Zhang et al [120]. Chapter 5 presents our conclusions and suggestions for future work.

Chapter 2

The ECDSA* scheme

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a standardized signature scheme [68]. It is the elliptic curve analog of the Digital Signature Algorithm (DSA), and is defined on a subgroup of an elliptic curve group over a finite field. ECDSA* is a modified version of ECDSA that represents signatures in uncompressed form. Uncompressed points are needed in order to *batch verify* ECDSA signatures. Batch verification of unmodified ECDSA signatures has been attempted in [74]; however the batch verification scheme was later shown to be insecure [24] (fixed in [71]).

The idea of batching operations was first considered by Fiat in the case of RSA [53]. Using this scheme, one can accelerate RSA signature verification by replacing full sized exponentiations with several smaller exponentiations with the same modulus N. Similarly, Naccache et al. proposed using random exponents to amortize the cost of modular exponentiations in DSA verification over the batch size [93]. In [18], Bellare extended these ideas by proposing batch verifiers for group exponentiations, namely the small exponents (SE) test and the bucket test (BT). The SE test is analagous to the idea from [93], while the BT performs SE on random buckets of a specified size. A recursive BT is only efficient for very large batch instances. Bernstein et al. note that the security levels of these early batch verification schemes are too low to be practical, and that improvement can be made in the exponentiation techniques [25].

The outline of the rest of this chapter is as follows. Section 2.1 describes the ECDSA* scheme. Section 2.2 gives an operation count for verifying a single ECDSA* signature. Section 2.3 presents a batch verification scheme by Cheon and Yi, and a detailed analysis for the 128-bit security level. Section 2.4 discusses a verification speedup for ECDSA* verification (which may be useful for batch verification).

2.1 Description

In what follows, $H : \{0,1\}^* \to \mathbb{Z}_n$ is a collision-resistant hash function, such as SHA-1. Recommended elliptic curve parameters for prime fields and characteristic 2 fields can be found in the NIST standard FIPS 186-4 [9]. In the next section, these choices are assumed.

Instantiation
Choose a large prime q. Choose constants $a, b \in \mathbb{F}_q$ to define

$$E/\mathbb{F}_q : y^2 = x^3 + ax + b, \tag{2.1}$$

and select a point $G \in E(\mathbb{F}_q)$ of prime order n.

The generation of parameters is done in a way that takes into account all of the standard attacks on the Elliptic Curve Discrete Logarithm Problem (ECDLP). Most importantly, the requirement that a large prime n divides $\#E(\mathbb{F}_q)$, with $(\log_2(n) \gtrapprox 160)$ ensures that the Pollard rho attack is infeasible [68]. Ideally, the cofactor $h = \#E(\mathbb{F}_q)/n$ should be small; for the NIST curves over prime fields, we have $h = 1$. The cost of this attack on two NIST curves is given in Table 2.1. There are special classes of elliptic curves where the ECDLP admits a faster solution than Pollard rho. For instance, the case where $\#E(\mathbb{F}_q) = q$ for q prime succumbs to the Araki-Satoh-Semaev-Smart attack [64], which takes linear time in the security parameter.

Curve	Cost for rho
NIST-P256	$2^{127.8}$
NIST-P384	$2^{191.8}$

Table 2.1: The cost of Pollard rho on NIST curves is approximately $\sqrt{\frac{\pi}{4}} \cdot 2^{n/2}$ point additions

We say that a signature scheme is *secure* if it is existentially unforgeable under chosen-message attack (cf. Definition 1.2). ECDSA security has not been proved equivalent to solving ECDLP, but it is clear that ECDLP must be intractable for key privacy. There has been some progress in proving security for ECDSA, but the results make strong assumptions [68]. We will assume that Pollard rho is the best attack on the signature scheme and the associated elliptic curve throughout this thesis.

Key Generation
The private key is $d \in_R [0, n-1]$.
The public key is $Q = dG$.

Signature Generation
Given message $M \in \{0,1\}^*$, compute the signature on M as follows.
Let $k \in_R [1, n-1]$, and $R = kG$. Let $r = x(R) \bmod n$.
Let $s = k^{-1}(H(M) + dr) \bmod n$.
Then (r, s) is the signature on M.

Signature Verification
A pair (r, s) of integers modulo n is a valid ECDSA signature on message M with public key Q if

$$R = (H(M) \cdot s^{-1} \bmod n)G + (rs^{-1} \bmod n)Q \tag{2.2}$$

satisfies

$$r = x(R) \bmod n.$$

In ECDSA*, we say that (R, s) is the signature on M, and only check that equation 2.2 holds.

It is preferable that we can continue to use ECDSA as it was standardized, and build functionality on top of it to gain efficiency. For this reason, Antipa et al. [11] proposed including side information with ECDSA signatures (r, s) on a message M so that the correct point R can be uniquely recovered, giving the corresponding ECDSA* signature (R, s) on the same message M. The candidate points for R come in pairs $\{R, -R\}$ since inversion in the group of elliptic curve points is just negation of the y-coordinate. In general, to go from r to the point R, we must "invert" the action of taking an x-coordinate modulo n. We are concerned exclusively with prime order curves in this chapter, so the cofactor $h = \#E(\mathbb{F}_q)/n$ is 1. Recall Hasse's Theorem: $|\#E(\mathbb{F}_q) - (q+1)| \leq 2\sqrt{q}$. Hence $q + 1 - 2\sqrt{q} \leq n \leq q + 1 + 2\sqrt{q}$. If $q < n$, then $r = x(R)$ with exact equality. If $n < q$, then we can deduce from Hasse's theorem that $q - n \leq 2\sqrt{q} - 1$. So there are at most $2\sqrt{q}$ elements of $\mathbb{Z}_q$ that are in multiple residue classes modulo n. The density of these elements is $\frac{q-n}{q} \leq \frac{2}{\sqrt{q}} - \frac{1}{q} \approx \frac{2}{\sqrt{q}}$, which is very small for q of practical interest. Since Hasse's theorem gives $(\sqrt{q} - 1)^2 \leq \sqrt{n}$, we have $q < 2n$. Otherwise, $q \geq 2n$ implies $n \leq \frac{1}{\sqrt{2}-1}$, a contradiction. So $x(R)$ can be r or $r + n$. Thus, we need one "sign" bit and one "magnitude" bit, which tells us which interval to map the residue mod n into. The general solution is to simply test the candidate points using the Antipa et al. accelerated verification procedure, which makes use of the same argument above. The authors of [11] note that in most cases, the overhead of testing candidates is still less than the gain from accelerating single verification. Of course, there is the option to use ECDSA* without modification for full efficiency, but we leave this to the needs of the application.

The following result from [11] states that there is no loss of security by using the modified signature scheme. However, as noted above, there is a small loss of efficiency.

Theorem 2.1 (ECDSA* Security [11])**.** *Consider ECDSA and ECDSA*. The following are equivalent:*
1. (r, s) *is a valid signature with respect to* (M, Q) *in ECDSA*
2. (R, s) *is a valid signature with respect to* (M, Q) *in ECDSA* for exactly one point* R *in the set* $\{P \in E(\mathbb{F}_q) \mid x(P) \bmod n = r)\}$

2.2 Speed-ups for Single Verification

Before considering fast batch verifiers, we want to compute the optimized cost of single verification. It is also useful to know whether these improvements are "orthogonal". In other words, is it possible to use the accelerated single verification to make the batch verifier even faster? We will answer this question and introduce some of the material necessary for operation counts in this chapter.

The optimization of Antipa et al. is enabled by storing precomputed multiples of the public base point G in the certificate for the public key Q, and thereafter performing half-size scalar multiplications. This reduces the number of doublings required by a factor of 2.

2.2.1 Method of Antipa et al.

Considering the verification equation (2.2), compute the coefficients

$$\begin{aligned} a &= H(M) \cdot s^{-1} \bmod n \\ b &= r \cdot s^{-1} \bmod n \end{aligned} \tag{2.3}$$

and then rewrite the verification equation as

$$aG + bQ - R = \mathcal{O}. \tag{2.4}$$

The idea presented in [11] is to multiply (2.4) by a scalar v such that the lengths of the new scalars will be a fraction of the length of the original scalars.

Theorem 2.2. *Let* $\alpha \in \mathbb{Z}$ *and let* B *be a positive integer. Then the congruence*

$$v\alpha \equiv u \bmod n \tag{2.5}$$

has an integer solution (u, v) *with* $|u| < n/B$ *and* $v \in [1, B]$.

Theorem 2.2 is a corollary of a result due to Dirichlet on Diophantine approximation. We will apply it assuming that $B < n$, where n is a large prime. This condition is needed for v to be invertible modulo n. Let $B = n^{1/2}$, so that the guaranteed solution satisfies

$$|u| < n^{1/2},\ 1 \leq v \leq n^{1/2}. \tag{2.6}$$

In equation 2.4, write the integer $b = (r \cdot s^{-1} \bmod n)$ as $b \equiv u/v \bmod n$, where u, v are as in Theorem 2.2. Then multiply (2.4) by v to obtain an equivalent verification equation

$$(a \cdot v \bmod n)G + uQ - vR = \mathcal{O}. \tag{2.7}$$

Since G is a public point, we assume that precomputation is available for this point. Thus, we can usefully decompose the scalar $\lambda = a \cdot v \bmod n$ into half-length scalars λ_0, λ_1 in the sense that we reduce the number of doublings needed to compute λG by a factor of 2:

$$\lambda G = \lambda_0 G + \lambda_1 (2^{\lceil t/2 \rceil} G). \tag{2.8}$$

Here, $t = \lceil \log_2 n \rceil$ is the bitlength of n and $\lambda = \lambda_0 + \lambda_1 2^{\lceil t/2 \rceil}$. We have split the scalar multiplication for G into two half-length scalar multiplications, given that we can precompute doublings of the base point G (and store them with the certificate). In other words, we have split λ into its most significant $t/2$ bits and its least significant $t/2$ bits. Substituting (2.8) for λG in (2.7), we have an equivalent verification equation

$$\lambda_0 G + \lambda_1 (2^{\lceil t/2 \rceil} G) + uQ - vR = \mathcal{O}. \tag{2.9}$$

Since the λ_i are half the bitlength of λ, and $|u|, |v| \leq n^{1/2}$ implies $\lceil \log_2 |u| \rceil, \lceil \log_2 |v| \rceil \leq \frac{1}{2}t$, every scalar is at most $t/2$ bits long. Thus computing the sum in (2.9) requires at most $t/2$ doublings by the interleaving method [64].

We will address the natural questions that follow:

1. How does one compute the integers u and v from Theorem 2.2?
2. Which strategy is optimal for the multi-scalar multiplication in (2.9)?
3. Assuming precomputed points are stored in the certificate and we use an optimal scalar multiplication strategy, what is the overall cost?

To compute the integers u and v, we can use the extended Euclidean algorithm (EEA). For the 2-scalar case, we obtain half-length coefficients during the execution of the EEA

on modulus n and the integer x (note that $\gcd(x, n) = 1$). That is, for the first remainder r_i in the sequence such that $r_i \leq \sqrt{n}$, we obtain v_i such that $c_i n + v_i x = r_i$ and $|v_i| \leq \sqrt{n}$. Reducing this equation mod n gives $x \equiv u/v$, as required. A proof that we can always efficiently compute a v_i with this property can be found in Section 4 of Gallant et al. [58].

The algorithm for 2-scalar multiplication used by Antipa et al. for (2.9) is the Joint Sparse Form (JSF), which we introduce next.

2.2.2 Joint Sparse Form

The Shamir-Strauss method is a technique for computing linear combinations of the form $aP + bQ$. A minimal working example for this method demonstrates that fewer curve additions are required than performing two ordinary scalar multiplications and adding the results. We first precompute and store $P + Q$. Then process the bits of a and b jointly from left to right, adding $P + Q$ whenever a '11' occurs in the joint binary representation, adding P when we encounter '10', adding Q for '01', and adding ∞ for '00'. For example, if

$$a = 0\ 1\ 0\ 1$$
$$b = 1\ 1\ 0\ 1$$

then

$$aP + bQ = (((\infty + Q)2 + P + Q)2 + \infty)2 + P + Q.$$

This requires a total of 3 doublings and 2 additions.

In general, we have scalars a and b of bitlength t. If the scalars

$$a = a_{t-1}\ a_{t-2}\ \cdots\ a_0$$
$$b = b_{t-1}\ b_{t-2}\ \cdots\ b_0$$

are chosen according to a uniform distribution, then the probability that $(a_i, b_i) = (0, 0)$ for a fixed i is $1/4$. So the probability that $(a_i, b_i) \neq (0, 0)$ is $3/4$. Thus, we expect $\frac{3}{4}t$ additions for random a, b. The number of doublings required is the bitlength of the scalars, so t doublings are needed. We can do better than this for the number of additions required.

The Joint Sparse Form (JSF) was invented by Solinas [112]. It uses the above left-to-right method, and it is more efficient than computing two independent scalar multiplications via 2-NAF (cf. Section 4 of [112]). Two independent NAFs require $5t/9$ additions on average, while the JSF requires $t/2$ additions. Although JSF requires an extra addition in the precomputation phase (i.e. $P + Q$ *and* $P - Q$), it remains advantageous to do this for all practical group orders and scalar lengths.

Definition 2.3. [64, 112]. Let

$$\begin{aligned} n_0 &= \langle u_{0,m}, \cdots, u_{0,1}, u_{0,0} \rangle = \sum_{i=0}^{m} u_{0,i} 2^i \\ n_1 &= \langle u_{1,m}, \cdots, u_{1,1}, u_{1,0} \rangle = \sum_{i=0}^{m} u_{1,i} 2^i \end{aligned} \tag{2.10}$$

be a signed radix-2 representation of n_0, n_1. A JSF of n_0 and n_1 has the following properties:

1. Of any three consecutive positions, at least one is a double zero.
2. Adjacent terms do not have opposite signs (the individual signed radix-2 representations are called 'reduced').
3. If $u_{i,j+1} u_{i,j} \neq 0$, then $u_{1-i,j+1} = \pm 1$ and $u_{1-i,j} = 0$.

There is an efficient algorithm algorithm for computing a JSF of two integers, and this representation can be shown to be unique.

2.2.3 Operation Count for Single Verification

This section gives the operation count for single verification in ECDSA*.

Let us give the explanation in the case where we precompute and store one multiple of the public key $Q = Q_0$, say $Q_1 = 2^{\lceil t/3 \rceil} Q$. Let $G_0 = G$, and let $G_1 = 2^{\lceil t/3 \rceil} G$ and $G_2 = 2^{2\lceil t/3 \rceil} G$ be precomputed multiples of G. Recall that the verification equation is

$$aG + bQ - R = \mathcal{O}. \tag{2.11}$$

Now, compute integers u, v such that $|u| \lesssim n^{2/3}, |v| \lesssim n^{1/3}$ and $vb \equiv u \mod n$ using the Extended Euclidean Algorithm, and multiply through by v in (2.11) to obtain

$$\lambda_0 G_0 + \lambda_1 G_1 + \lambda_2 G_2 + uQ - vR = \mathcal{O}, \tag{2.12}$$

where λ_0, λ_1, and λ_2 are the three limbs of length approximately $t/3$ of the scalar $av \mod n$. Since $|u| \lesssim n^{2/3}$, we split u into two limbs of roughly $t/3$ bits each by letting $u_0 = u \mod 2^{t/3}$ and $u_1 = (u - u_0)/2^{t/3}$. The result is a verification equation in which all scalars have size $\lesssim t/3$ bits:

$$\lambda_0 G_0 + \lambda_1 G_1 + \lambda_2 G_2 + u_0 Q_0 + u_1 Q_1 - vR = \mathcal{O}. \tag{2.13}$$

We now group the scalar multiplications in pairs and find $\mathrm{JSF}(\lambda_0, \lambda_1), \mathrm{JSF}(\lambda_2, u_0)$, and $\mathrm{JSF}(u_1, v)$. The remaining task consists of three independent 2-scalar multiplications (and adding the results).

Thus, the overall cost of single verification is approximately

$$\left[3\left(\frac{1}{2}\cdot\frac{t}{3}+2\right)+2\right]\mathcal{A}_e+\left(\frac{t}{3}\right)\mathcal{D}_e=(t/2+8)\mathcal{A}_e+(t/3)\mathcal{D}_e, \tag{2.14}$$

where $\mathcal{A}_e$ denotes elliptic curve additions and $\mathcal{D}_e$ denotes doublings. We use the term 'approximately' here because the exact number of additions required depends on the JSF of the integers involved in the scalar multiplications; we are working with expected values. Also, the JSF may be 1 digit longer than the respective binary expansions.

Coordinates

In order to express equation 2.14 in terms of field operations – inversions, multiplications and squarings – a choice of *efficient* coordinates must be made. The affine addition formulas that define the group law are not the most efficient to use in practice when we have a large number of points to add, due to the necessary field inversions. Standard projective coordinates are the first step in eliminating the costly inversions associated with affine coordinates. We include these standard definitions from [64]. Note that we work over the finite field $\mathbb{F}_q$.

Definition 2.4. Let c, d be positive integers. Let a relation $\sim$ be defined on $\mathbb{F}_q^3\backslash\{(0,0,0)\}$ by $(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2)$ if and only if $X_1 = \lambda^c X_2,\ Y_1 = \lambda^d Y_2,\ Z_1 = \lambda Z_2$ for some $\lambda \in \mathbb{F}_q\backslash\{0\}$. The equivalence class $(X : Y : Z)$ consists of all triples that are related by $\sim$. We call $(X : Y : Z)$ a **projective point**.

The associated projective form of the Weierstrass equation 2.1) is obtained by substituting $(X/Z^c, Y/Z^d)$ for (x, y) and clearing denominators. The same substitution is applied to obtain projective doubling and addition formulas; we simply carry an extra third coordinate Z throughout computation, which can be thought of as an accumulated denominator. This is helpful because the group law in projective coordinates requires no inversion. We can later convert back to affine coordinates as needed by inverting this third coordinate. However, note that this inversion only needs to be done once per bulk operation, e.g., once per scalar multiplication, or once per multi-scalar multiplication.

In the case where $c = d = 1$, the equivalence classes are said to be in **standard projective coordinates**. We will assume that the curve parameter a is -3, as the work

of Brier and Joye shows that we do not lose much generality [36]. [1] The projective equation of the elliptic curve in these coordinates is

$$Y^2Z = X^3 + aXZ^2 + bZ^3. \tag{2.15}$$

The distinguished element $(0 : 1 : 0)$ corresponds to the point at infinity on E.

It turns out that we can do slightly better by choosing different c, d. In the case where $c = 2, d = 3$, projective points are said to be in **Jacobian coordinates**. The projective equation of the elliptic curve in these coordinates is

$$Y^2 = X^3 + aXZ^4 + bZ^6. \tag{2.16}$$

The point at infinity ∞ corresponds to the projective point $(1 : 1 : 0)$. The formulas for doubling a point $P = (X_1 : Y_1 : Z_1)$ in Jacobian coordinates are given here:

$$\begin{aligned} X_3 &= (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2 \\ Y_3 &= (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4 \\ Z_3 &= 2Y_1Z_1 \\ 2P &= (X_3, Y_3, Z_3). \end{aligned}$$

Note that since $a = -3$, we can rewrite a common subexpression

$$3X_1^2 + -3Z_1^4 = 3(X_1 - Z_1^2) \cdot (X_1 + Z_1^2).$$

This new expression can be computed in 1 field squaring and 1 field multiplication since multiplication by a small constant is counted as very fast, and additions mod q are negligible in comparison to multiplications mod q. This explains the savings of two squarings over the general case: we trade multiplication by a for multiplication of $(X_1 - Z_1^2)$ and $(X_1 + Z_1^2)$, and three squarings for 1 squaring of Z_1. We also note that in order to count operations precisely, we have to divide the above formulas into computed units and give the full sequence of operations. This is standard and can be found in [64]. Further optimizations are possible but we will not go into further detail. We summarize the curve operation counts in Table 2.2.

[1] We also gain efficiency, for example, general doubling in Jacobian coordinates costs $4m + 6s$, and letting $a = -3$ reduces this to $4m + 4s$.

Doubling		General addition		Mixed coordinates	
$2A \to A$	$1i + 2m + 2s$	$A + A \to A$	$1i + 2m + 1s$	$J + A \to J$	$8m + 3s$
$2P \to P$	$7m + 3s$	$P + P \to P$	$12m + 2s$		
$2J \to J$	$4m + 4s$	$J + J \to J$	$12m + 4s$		

Table 2.2: Operation counts for curves over prime fields [64]

2.3 Batch Verification

In this section we present two batch verification methods for ECDSA*. First, we discuss why it is important to be careful: a batch verifier should not allow forgeries to pass verification e.g., via cancellations in the multiscalar multiplication. Karati et al. [74] proposed a batch verification scheme which uses only original ECDSA signatures. The advantage of this is a smaller signature size, since if $\lceil \log_2(q) \rceil = t$, then ECDSA has signatures of size $\approx 2t$ whereas ECDSA* has signatures of size $\approx 3t$. As we mentioned before, there is a small amount of overhead for testing candidate points, but it is not a deciding factor in practice between these two schemes. We can still have small signature size by doing point recovery in the standard way (square roots to recover y-coordinates) per-signature for compatibility reasons and flexibility. The main benefit of ECDSA* is working in the cyclic group generated by a base point, and forming linear combinations. Constructing batch verifiers using these ideas has great potential for efficiency enhancements in ECC protocols. However, if we demand a small signature size, then the cost of decompression (i.e. computing modular square roots) must be considered. There has been work done on reducing the overhead associated with decompression, but it turns out that weaknesses can arise. We follow a paper of Bernstein et al. [24] that discusses an example.

2.3.1 Importance of Random Scalars

Consistent with the idea of batching, Karati et al. [74] combine all the individual ECDSA* verification equations into one equation via addition. Looking at equation 2.4, we see that a summation $\sum_{i=0}^{N} R_i$ of signature points arises; the technical work presented in the paper presents new techniques for x-coordinate only computation of this sum. Their proposed verification equation in that work is the following (computation of the left hand side is the novelty of their work):

$$\sum_{i=0}^{N} R_i = \left(\sum_{i=0}^{N-1} H(M_i) s_i^{-1} \right) G + \sum_{i=0}^{N-1} (r_i s_i^{-1}) Q_i. \tag{2.17}$$

Despite the gain in efficiency over individual verification, there is a way to make forgeries pass this verification equation, even though they would fail individual verification. A batch verifier must ensure that the probability of a forgery (incorrect instance) being accepted is exponentially small in the security parameter [18]. Bernstein showed that this is not the case for this batch construction. We note that Karati et al. generalized their methods to the case of random scalars in [70] to respond to this attack. The basic fix is to use symbolic methods for batch size 2, and a Montgomery ladder for batch size 3 to 8; larger batch sizes are not considered due to higher cost than single verification. It is not clear what effect inclusion of random scalars has on performance relative to the original claims.

The attack

Consider an attacker with private key a_2 and corresponding public key Q_2. Bernstein et al. showed that the attacker can make any message M_1 pass batch verification under *any target public key* Q_1, along with a message M_2 signed by Q_2. Of course, both messages will still pass verification in larger batches due to the cancellation that occurs in a non-randomized sum of curve points.

The attacker picks a random $k_1 \in_R [1, n-1]$ and computes $R_1 = k_1 P$ and corresponding $r_1 = x(R_1) \mod n$ as in proper signing. Then it chooses $s_1 \in_R [1, n-1]$, which is *not* what is done in proper signature generation – the integer s_1 should be derived from the private key, the hash of the message, and the randomly chosen k_1.
For the second message-signature pair, the attacker computes $R_2 = \left(\frac{r_1}{s_1}\right) A_1$, which is not a random multiple of the base point G as in proper signing. Then it computes $r_2 = x(R_2) \mod n$ and

$$s_2 = \frac{(H(M_2) + r_2 a_2)}{\left(k_1 - \frac{H(M_1)}{s_1}\right)}. \tag{2.18}$$

The verifier receives $(r_1, s_1), (r_2, s_2)$ as purported signatures on M_1, M_2 as in the ordinary protocol. It then reconstructs the full points R_1, R_2 via square root computation and subsequently $R_1 + R_2$, or directly from 'almost' x-coordinates r_1, r_2 with Karati et al.'s symbolic methods. Then it computes the right hand side (how is not important here)

$$\left(\frac{H(M_1)}{s_1} + \frac{H(M_2)}{s_2}\right) G + \left(\frac{r_1}{s_1}\right) Q_1 + \left(\frac{r_2}{s_2}\right) Q_2. \tag{2.19}$$

Substituting the choice of s_2 and simplifying this expression gives $\frac{r_1}{s_1} Q_1 + k_1 G$, and since $R_1 + R_2 = k_1 G + (r_1/s_1) Q_1$, these two sides are equal. Thus batch verification containing

these two signatures passes for any message, and any choice of public key A_1. Since the choice of s_1 is random, individual verification fails with overwhelming probability, since $s_1 \neq k_1^{-1}(H(M_1) + a_1 r_1) \mod n$ with high probability, and thus the right hand side $\frac{H(M_1)}{s_1}G + \frac{r_1}{s_1}a_1 G$ will not factor to give $R_1 = k_1 G$. Similarly, we can check that (r_2, s_2) on M_2 will not pass single verification by substituting the above choices.

Given this attack, it is clear that inclusion of random scalars in batch verification is needed.

2.3.2 Cheon-Yi

Another idea for batch verification of ECDSA* signatures is to generalize the ideas of Bellare et al. [18] and Cheon/Lee's sparse exponents [41]. The construction of Cheon and Yi [42] takes into account the attack of Boyd and Pavlovski [35] that applied to some early schemes that did not correctly adapt the small exponents test. The small exponents test (Algorithm 1) is a batch verifier for a collection of exponentiations in a group of prime order q. Note that this test assumes the points y_i are in G. Cheon and Yi remark that in order to defeat/ignore completely the small subgroup attack that Boyd and Pavlovski propose, $E(\mathbb{F}_q)$ should have cofactor 1.

Algorithm 1 Small Exponents Test

INPUT: g a generator of G, and $(x_1, y_1), ..., (x_N, y_N)$ with $x_i \in \mathbb{Z}_n$ and $y_i \in G$, security parameter ℓ

CHECK: That $y_i = g^{x_i}$ for all $i \in \{1, ..., N\}$

(1) Pick $s_1, ..., s_N \in \{0,1\}^\ell$ at random.

(2) Compute $x = \sum_{i=1}^{N} x_i s_i \mod n$, and $y = \prod_{i=1}^{N} y_i^{s_i}$.

(3) If $g^x = y$ then accept, else reject.

Modifications are necessary for other cofactors (e.g. 2 and 4 for the NIST curves defined over characteristic-two fields), otherwise we have to consider those curves weak since forgeries of the form $\alpha \cdot y_i$ can pass through verification. The modification proposed to the small exponents test is to "give assurance of the correctness of the batch up to multiplication by an element of order less than 2^ℓ". This means that αy_i will be considered valid for any element α of small order in the group.

We emphasize that Algorithm 1 and the corresponding proof of at most $2^{-\ell}$ error probability are for single exponentiations in a group. Also note that Algorithm 1 is generic and leaves open the methods of computation, the optimal choice of which depends highly on

the group and the cryptographic protocol we work with. One thing we can say immediately is that the computation of the product $\prod_{i=1}^{N} y_i^{s_i}$ is easier in a group where inversion is free, e.g. elliptic curve groups, since we can use $\mathrm{NAF}(s_i)$.

Cheon and Yi propose to do ECDSA* batch verification in the usual fashion: verify a random linear combination of the ECDSA* verification equations arising from a batch instance. Their approach chooses the scalars to be integers in width-w non-adjacent form (w-NAF). A w-NAF is a signed radix-2 representation of an integer in which every nonzero digit is an odd integer with absolute value less than 2^{w-1}, and at most one of any w consecutive digits is nonzero. For the purposes of their work, we also require that the number of nonzero digits be a chosen parameter t. The associated odd-digit set is $\mathcal{D} = \{\pm 1, \pm 3, ..., \pm(2^{w-2} - 1)\}$. The length of the representation is the index of the most significant digit; the weight of the representation is the number of nonzero digits.

First, we must generate the random scalars in w-NAF form. There are many radix-2 representations of an integer, so we must be sure that the random scalars we generate are uniformly distributed, i.e. they are distinct as integers, so that the subsequent enumeration argument (Lemma 2.6) is valid. The sufficient condition for this is the following:

Lemma 2.5. [42] Let n be a positive integer. All w-NAFs of length $\leq m$ with the digit set $\mathcal{D}$ are distinct modulo n if $m \leq \log_2(n/C)$ where $C = \max\{|x - y| : x, y \in \mathcal{D} \cup \{0\}\}$.

In the case of elliptic curves with $\mathcal{D} = \{\pm 1, \pm 3, ..., \pm(2^{w-2} - 1)\}$, we have $C = 2^{w-1} - 2$, and so we need $m \leq \log_2(n/(2^{w-1} - 2)) \approx \log_2 n - w + 1$. Thus we require that $m \leq \log_2 n - w + 1$. Subject to this condition, we now count the number of possible w-NAF scalars with these parameters in order to establish the security level of the batching scheme.

Lemma 2.6. [42] The number of w-NAFs of length $\leq m$ and weight t with the digit set $D = \{\pm 1, \pm 2, ..., \pm(2^{w-2} - 1)\}$ is

$$\binom{m - (w-1)(t-1)}{t} 2^{(w-1)t}.$$

The proof of this lemma yields Algorithm 2 for generating random w-NAFs for batch verification. For a choice of the parameters m, t, w, this is the security level of the batching scheme. Now suppose we are given N ECDSA* signatures (m_i, R_i, s_i), which correspond to signers with public keys Q_i. Letting $c_1, ..., c_N$ be randomly chosen w-NAFs via Algorithm 2, we have to compute

$$\sum_{i=1}^{N} c_i(a_i G + b_i Q_i + R_i) = aG + \sum_{i=1}^{N} b_i' Q_i + \sum_{i=1}^{N} c_i R_i \tag{2.20}$$

Algorithm 2 Generation of w-NAF exponents of weight t

INPUT: m, w, t and the digit set $\mathcal{D}$

OUTPUT: w-NAF of length $\leq m$ and weight t

(1) Choose t positions out of $m - (w-1)(t-1)$ positions

(2) Fill each position by $(w-1)$ consecutive zeros followed by an element in $\mathcal{D}$

(3) Discard the first $(w-1)$ positions of the string

(4) Print the string which is a w-NAF of length $\leq m$

where $a_i = -H(M) \cdot s_i^{-1} \bmod n$, $b_i = -r_i \cdot s^{-1} \bmod n$, $a = \sum_{i=1}^{N} c_i a_i \bmod n$, and $b'_i = c_i b'_i \bmod n$. In the multiple signer case, we assume that all Q_i are distinct, and in the single signer case, we have $Q_i = Q$ for all i. So we consider two strategies, accordingly. The

	single signer	multiple signers
equation	$aG + bQ + \sum_{i=1}^{N} c_i R_i$	$aG + \sum_{i=1}^{N} b'_i Q_i + \sum_{i=1}^{N} c_i R_i$
cost	1 JSF + N (w,t)-NAFs	$(N+1)$ w-NAFs + N (w,t)-NAFs

Table 2.3: Counting batch operations – the single signer and multiple signer cases.

two cases are considered in Table 2.3. Note that the signature points R_i have corresponding scalars chosen according to Algorithm 2 but the public keys Q_i have essentially random scalars, so we are free to re-code them in the most efficient way. Also note that the parameter w may be different in each case, since it is selected based on the amount of storage available and the application scenario.

Assuming a single signer means we could approach single verification as two fixed-base scalar multiplications. The algorithm used in this case is called the "fixed base comb". This method uses a moderate amount of precomputation to reduce the number of doublings. However, the amount of precomputation per point makes it perform no better than (2.14). Thus, we do not distinguish between single and multiple-signer verification when not batching.

Now, to obtain operation counts for the verifications in Table 2.3 we can make use of 'interleaving' [64]. In this method, the individual scalars k_i in a multi-scalar multiplication $\sum_i k_i P_i$ are re-coded using possibly different methods (e.g. w-NAF for various w). There is a single accumulator for additions per inner loop iteration, and doubling is done jointly. A consequence of the latter is that the number of doublings is the maximum length of a scalar k_i in the instance, so the benefit of doing fixed-base comb will be lost.

In the single-signer case, we have one joint sparse form 2-scalar multiplication, together with a sum of w-NAFs arising from the generation Algorithm 2. We let $m = 252$, $w = 5$,

$t = 14$, giving a security level of $\approx 2^{125.99}$. Notice that $252 = m \leq 256 - w + 1 = 252$, so uniqueness of the scalars is guaranteed. Since we have to compute w-NAFs of unknown points, online precomputation for multiples in $\mathcal{D}$ of the R_i is needed. The cost of the multi-scalar multiplication is

$$N \cdot (2^{w-2}\mathcal{A}_e + 1\mathcal{D}_e) + N \cdot t\mathcal{A}_e + m\mathcal{D}_e. \tag{2.21}$$

The above count is split into online precomputation and actual computation, respectively. Substituting our choices for the parameters, we obtain the following approximate ratio of batch-to-single verification

$$(N(22\mathcal{A}_e + \mathcal{D}_e) + 252\mathcal{D}_e)/N(136\mathcal{A}_e + 86\mathcal{D}_e).$$

For a more precise comparison, we should convert these estimates using the field operation counts from Table 2.2. Using $\mathcal{A}_e = 8m + 3s$, $\mathcal{D}_e = 4m + 4s$, and $s = m$, the above ratio is

$$0.114 + 0.92/N$$

Thus, it is clear that for any batch size (at least two), there is a clear advantage for batching in the single signer case.

For the multiple signer case, since we use interleaving there is no reason to use a fixed base comb for G; thus, we have $(N + 1)$ random w-NAFs to compute, and N w-NAFs generated with weight exactly t. Note that the scalars corresponding to the Q_i have average weight $m/(w + 1) = 256/6 = 42.7$, while the c_i have weight $t = 14$. Since we are using interleaving, we need 256 doublings in this case. The total cost of batching for multiple signers is thus

$$\left(\frac{256}{w+1}\mathcal{A}_e\right) + 2N(2^{w-2}\mathcal{A}_e + 1\mathcal{D}_e) + Nt\mathcal{A}_e + N\frac{256}{6}\mathcal{A}_e + 256\mathcal{D}_e \tag{2.22}$$

$$= \frac{128}{3}\mathcal{A}_e + 256\mathcal{D}_e + 72.7N\mathcal{A}_e + 2N\mathcal{D}_e. \tag{2.23}$$

Using the same doubling-to-addition ratio as in Section 2.2.3, we obtain

$$228.88\mathcal{A}_e + 74.15N\mathcal{A}_e. \tag{2.24}$$

This gives a ratio of batch verification to single verification cost of

$$0.373 + 1.15/N.$$

We again obtain the same conclusion as for the single signer case: for large batch sizes, there is an appreciable speed up of signature verification. Note that due to the contributions of doublings and precomputations, larger batch sizes are preferred to obtain maximum benefit, whereas smaller batch sizes are less attractive.

Hakuta et al. [63] propose an interesting batch verification scheme for special fields of characteristic 5 and 7 which are more efficient than Cheon and Yi's complex exponent test (which we have just discussed) on small batches at the $\approx$ 128-bit security level. Cheon and Yi discuss how to optimize batch verification for binary curves as well, but we do not consider this any further.

Recall the question we posed at the beginning of Section 2.2: can we combine the technique of Antipa et al. with Cheon-Yi batching to attain even faster verification? There is a generalization of the method for simultaneous reduction in scalar size for N scalars (see Example 7 of [11]). This reduces all but one exponent to $1-\epsilon$ size, but requires finding a short vector in an $N+1$ dimensional lattice. Thus, the technique is efficient for small batch sizes but would require rigorous testing to demonstrate that it can be efficient for larger batch sizes (e.g. $N = 64$). Moreover, it seems likely that the lattice computation would undo the sparseness of the scalars in the Cheon-Yi method, which have a fixed length representation in order to guarantee a chosen security level. Thus, in the case of ECDSA* the answer seems to be that these two methods are orthogonal.

2.4 Acceleration through Caching

Möller and Rupp [90] consider practical scenarios in which ECDSA can be made more efficient when performing multi-exponentiation during signature verification. The novelty in the approach comes from storing powers of a base that happen to arise without extra effort if the computation is arranged suitably. The basic idea is to consider multi-exponentiation problems

$$\prod_{1 \leq i \leq k} g_i^{e_i}.$$

In the case of ECDSA where $k = 2$, we can make some assumptions on how often we see the bases g_i and exponents e_i, as well as the amount of precomputed data we can store. The authors assume that read-only-memory is not severely limited, so that precomputation on $g_2, ..., g_k$ can be stored permanently.

There are three algorithms used in this technique: interleaving for multi-exponentiation, exponentiation and caching for a new base g_1, and exponentiation for an old base g_1, i.e.

we have found this base in the cache, and we want to use old computed intermediate results in a new verification computation.

Algorithm 3 Interleaving

INPUT: Precomputed g_i^b for all $b \neq 0 \in B_i$ signed digit sets; exponents $e_i = (b_{i,\ell}, b_{i,\ell-1}, ..., b_{i,0})$
OUTPUT: $\prod_{i=1}^{k} g_i^{e_i}$

```
A ← 1_G
for j = ℓ...0 do
    A ← A^2
    for i = 1...k do
        if b_{i,j} ≠ 0 then
            A ← A · g_i^{b_{i,j}}
        end if
    end for
end for
return A
```

This is the familiar interleaving algorithm discussed in the context of batching in Section 2.3.2, but in this case we have a single multi-exponentiation in mind.

Now, we split the exponent of base g_2 into limbs of roughly equal lengths $L_s, ..., L_1 \approx \frac{\ell+1}{s}$

$$e_2 = (b_{2,L_1+L_2+\cdots+L_s-1}, \ldots, b_{2,L_1+\cdots+L_{s-1}}, \ldots, b_{2,L_1-1}, \ldots, b_{2,0}).$$

We require that $L_1 + 1 \geq \max_{1 \leq i \leq s} L_i$. In the simplest case, we can split the exponents into two limbs.

Suppose that a new g_1 is encountered. Now, after splitting the exponent for g_2 (new g_1's are never split: we have no precomputed data on them by definition of being new), we have transformed the problem into a multi-exponentiation with bases

$$(g_1, g_2, g_2^{2^{L_1}}, \ldots, g_2^{2^{L_1+\cdots+L_{s-1}}})$$

and corresponding exponents

$$\{e_1, (b_{2,L_1-1}, b_{2,L_1-2}, \ldots, b_{2,0})_2, \ldots, (b_{2,L_1+\cdots+L_s-1}, \ldots, b_{2,L_1+\cdots+L_{s-1}})_2\}.$$

Now, the exponents of the powers of g_2 all have maximum length $L_1 + 1$, and the exponent of g_1 is *full-length*, i.e. $\ell = L_1 + \cdots + L_s$ bits long. In particular, looking at Algorithm 3,

no iteration before $j = L_1$ can have a nonzero exponent bit for any of the powers of g_2, and so all iterations before involve exponentiations of g_1. By making this observation, we save g_1, along with intermediate powers of g_1:

$$\{g_1, [(b_{1,l}, \ldots, b_{1,L_1})_2, g_1^{(b_{1,l},\ldots,b_{1,L_1})_2}]$$
$$\vdots$$
$$[(b_{1,l}, \ldots, b_{1,L_1+\cdots+L_{s-1}})_2, g_1^{(b_{1,l},\ldots,b_{1,L_1+\cdots+L_{s-1}})_2}]\}.$$

Note that the exponents start out long and become shorter in the above list of cached data. For multi-exponentiation where an old base g_1 is found in cache, we re-use cached data using the following algorithm. We denote by $G_i = g_1^{\lambda_i}$ for $1 \leq i \leq s-1$ the precomputed powers of g_1, and $\lambda_1 \geq \ldots \geq \lambda_{s-1} \geq 0$. Algorithm 4 allows us to write e_1 in terms of

Algorithm 4 Modular Exponent Splitting

INPUT: Cached data $(g_1, (\lambda_1, G_1), (\lambda_2, G_2), \ldots, (\lambda_{s-1}, G_{s-1}))$
OUTPUT: Exponents E_i

$d_0 \leftarrow e_1$
for $1 \leq i \leq s-1$ **do**
  $E_i \leftarrow \lfloor \frac{d_{i-1}}{\lambda_i} \rfloor$ and $d_i \leftarrow d_{i-1} - E_i\lambda_i$
end for
$E_s \leftarrow d_{s-1}$
return $E_1, \ldots, E_s$

exponents that we have computed powers of g_1 for:

$$e_1 = E_1\lambda_1 + \cdots + E_{s-1}\lambda_{s-1} + E_s.$$

It is easy to check that this holds by definition of the d_i. Now, the exponent for g_2 is split according to the fixed L_i in radix-2 exponent splitting, and we perform interleaving on the resulting ks-fold multi-exponentiation problem. Note that in the case $s = 2$, we have split the exponents in half.

2.4.1 Operation Count

We present an example similar to the one in Appendix C of [90] to demonstrate how this technique is useful. At a high level, we are simply re-using exponentiations of g_1, assuming

that we can store these powers. An example of processing exponents in this way can be found in Appendix A of [90].

In the case of ECDSA with a prime order Weierstrass curve at the 128-bit security level, we deal with exponentiation of the form $g_1^{e_1} g_2^{e_2}$. The base element g_2 is the fixed base point on the curve, and g_1 is the variable public key. This public key will differ from the last with some probability, P_{old}, which depends on the application. Assume we have enough read/write memory to store two group elements, as well as the temporary variable A in Algorithm 3, and the 256-bit exponents e_1, e_2.

If we have a new base g_1, we precompute g_1^3 and keep g_1, g_1^3 in read/write memory while converting $(b_{1,255}, \ldots, b_{1,128})_2$ to signed digit representation *on the fly*, with digit set $\{0, \pm 1, \pm 3\}$. As soon as exponent conversion is done and we have *computed* $g_1^{\lambda_1}$ (which costs $128/(2+2)m + 127s$) we store the result in read/write memory in the place where g_1^3 was stored previously. We then use digit set $\{0, \pm 1\}$ for the lower order bits $(b_{1,127}, \ldots, b_{1,0})_2$ and perform the final interleaving algorithm. The removal of g_1^3 is being done here because of our assumption about available read/write memory. The estimated cost in field multiplications is (compute $g_1^{\lambda_1}$) + (compute interleaving with powers of g_2)

$$\left(\frac{128}{2+2}\right) m + 127s + \left(\frac{128}{2+1} + 2 \cdot \frac{128}{2+7}\right) m + 127s + 1s \approx 281.6m.$$

If we encounter an old base g_1, then we load the cached power of g_1, namely $g_1^{\lambda_1} = G_1$, and we have

$$g_1^{e_1} g_2^{e_2} = g_1^{E_1 \lambda_1 + E_2} g_2^{e_2} = G_1^{E_1} \cdot g_1^{E_2} \cdot g_2^{(b_{2,255}, \ldots, b_{2,128})_2} \cdot g_2^{(b_{2,127}, \ldots, b_{2,0})_2}.$$

Again, we only have digit set $\{0, \pm 1\}$ for exponentiation with base g_1, since we use a cached entry. Perform interleaving and assume that E_2 results in an exponent of approximately the same length as E_1 (the authors of [90] mention that if e_1 is uniformly random, then the maximum length of E_i will remain around $L \approx (l+1)/s$ with high probability). This costs

$$\left(\frac{128}{2+1} + \frac{128}{2+1} + 2 \cdot \frac{128}{2+7}\right) m + 127s \approx 202.7m.$$

It is easy to show that the cost of simple interleaving with signed digit sets $\{0, \pm 1, \pm 3\}$ for g_1 and $\{0, \pm 1, \ldots, \pm 8\}$ for g_2 costs approximately $268.1m$, so depending on the probability P_{old} of public key repetition, the average cost of verification using caching and modular exponent splitting is

$$202.7m \times (1 - P_{old}) + 281.6m \times P_{old}.$$

For $P_{old} = 0.5$ this is $242.1m$, an approximate 10% reduction in modular multiplications.

We note that the Möller-Rupp caching technique can be used in batch verification when public keys re-occur. A larger amount of memory is needed, however, since larger batch sizes require at least as much space for storing group elements (768 bits for points in projective coordinates). It would be interesting to estimate the cost of verifying batch instances with some repeated public keys, assuming we have enough cache.

Chapter 3

High-Speed High-Security Signatures

In 2006, Bernstein proposed the use of curve25519 for use as an efficient and secure elliptic curve Diffie-Hellman function [20]. Curve25519 is expressed in Montgomery form [91]. After the discovery of the Edwards form of curves [50], a series of projective coordinate systems were proposed for implementations using this form [22, 67]. These improvements were used in the 2012 proposal of Bernstein, Lange, Schwabe, Yang, and Duif [25] for high-speed high-security signatures that make use of batch verification. The implementation also uses the prime $2^{255} - 19$ for fast field arithmetic, as in curve25519, and a twisted Edwards curve (which is birationally equivalent to curve25519).

The outline of this chapter is as follows. Section 3.1 presents a proof of security for the Schnorr signature scheme. Section 3.2 introduces Edwards curves and related background. Section 3.3 describes the EdDSA signature scheme. Section 3.4 gives point addition algorithms for various coordinate systems derived for Edwards curves. Finally, Section 3.5 gives background on batch verification with the Bos-Coster algorithm, and an alternate derivation for the cost of the algorithm. We also give operation counts for generic Bos-Coster at the 128 and 256-bit security levels.

3.1 Schnorr Signatures

In this section we consider the Schnorr signature scheme. It is a variant of the well-known ElGamal signature scheme, set in a prime-order subgroup of the multiplicative group of a finite field. It will also serve as a basis for comparison with EdDSA. The following description of Schnorr signatures is from Stinson [114].

Let p be a (large) prime such that the discrete logarithm problem in $\mathbb{Z}_p^*$ is intractable, and let q be a prime that divides $p-1$. Let $g \in \mathbb{Z}_p^*$ be a qth root of unity modulo p. Let $\mathcal{P} = \{0,1\}^*$ be the message space, $\mathcal{A} = \mathbb{Z}_q \times \mathbb{Z}_q$ the signature space, and define

$$\mathcal{K} = \{(p, q, g, x, g^x) : x \in \mathbb{Z}_q\}$$

to be the set of public, private key tuples. The values p, q, g, g^x are public, and x is the private key. Finally, let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a secure hash function.

For $K = (p, q, g, a, g^x)$, and for a (secret) random number k, $1 \leq k \leq q-1$, define

$$\mathbf{sig}_K(M, k) = (h, s)$$

where the signer computes $r = g^k \bmod p$,

$$h = H(M||r),$$

and

$$s = k + xh \bmod q.$$

For $M \in \{0,1\}^*$ and $h, s \in \mathbb{Z}_q$, verification is done by performing the following computations:

$$\mathbf{ver}_K(M, (h, s)) = \text{true} \Leftrightarrow H(M||g^s(g^x)^{-h} \bmod p) = h.$$

The Schnorr signature scheme enjoys the property of being provably unforgeable under adaptive chosen-message attack. This was initially surprising, since textbook RSA and ElGamal signatures succumb to existential forgery. The original paper by Schnorr gave a proof of security for the related Schnorr identification scheme. A security proof for Schnorr signatures was later given by Pointcheval and Stern [96]. In this thesis, we will follow the reductionist security argument given by Koblitz and Menezes [78]. The argument shows that, subject to a loss of efficiency, a chosen-message existential forger can use his power to compute discrete logarithms (that is, the discrete logarithm of the public key).

Since we deal with randomized algorithms in this section, we give the necessary definitions before the security proof.

Definition 3.1 (PPT [60])**.** A **probabilistic polynomial time algorithm** is an algorithm $\mathcal{A}$ such that:

1. there is an arbitrarily long stream of purely random bits available to $\mathcal{A}$, considered as auxiliary input to a deterministic algorithm

2. there exists a polynomial f such that, for any input of size ℓ, the algorithm $\mathcal{A}$ terminates after $f(\ell)$ steps, and outputs a correct answer with probability at least $1-\epsilon(\ell)$, where the probability is taken over the distribution of random bits accessed by $\mathcal{A}$. Here, ϵ is a negligible function.

A consequence of part 1 of Definition 3.1 is that the algorithm is in some sense deterministic: if it made the same sequence of random choices, the output would always be the same. A consequence of part 2 is that a PPT algorithm may give an incorrect output for some "unlucky" random choices, but it succeeds for most choices. In general a PPT algorithm may also fail to produce an answer, but for our purposes this is the same as giving an incorrect answer. Finally, we note that part 2 implies that the algorithm must always *terminate*, and do so in polynomial time.

Definition 3.2 (Success Probability [57])**.** Let $\epsilon : \mathbb{N} \to \mathbb{R}_{>0}$ be a function. Then ϵ is **negligible** if for every polynomial $p(x) \in \mathbb{R}[x]$ there is some $K \in \mathbb{N}$ such that for all $\kappa > K$ with $p(\kappa) \neq 0$ we have $\epsilon(\kappa) < 1/|(p(\kappa)|$.

The reductionist security argument makes use of two lemmas. The "splitting lemma" is used to count the number of "good" pairs in a cartesian product of sets. The "forking lemma" will be introduced in the course of the proof, and it has to do with giving a new, independent sequence of random bits, and independent, random function values to a copy of the forger oracle. Then, if both forger copies produce a forgery on a chosen message, we can compute a discrete logarithm.

We use the notation from [78], since we lose no generality in the generic group setting, and this proof does not require the setting of $\mathbb{Z}_p^*$.

Theorem 3.3 (Splitting Lemma)**.** *Let A, B be finite sets with $\#A = a, \#B = b$. Suppose that ϵab of all pairs $(\alpha, \beta) \in A \times B$ are "good". Let $A_0 \subset A$ be defined as the set of elements α_0 such that a pair (α_0, β) (with $\alpha_0 \in A$ fixed and $\beta \in B$ varying) has probability $\geq \epsilon/2$ of being "good". Then there are at least $\epsilon a/2$ elements in A_0.*

Proof. Assume, towards a contradiction, that $\#A_0 < \epsilon a/2$. We count the number of good pairs with $\alpha \in A_0$ and the number of good pairs with $\alpha \notin A_0$.

The number of good pairs with $\alpha \in A_0$ is at most $\#A_0 \cdot b < \epsilon ab/2$. The number of good pairs with $\alpha \notin A_0$ is $< \#(A \backslash A_0) \cdot \#B \cdot \epsilon/2$, since by definition of A_0, a pair (α_0, β) is in $A_0 \times B$ if and only if the probability of any pair in the α_0 row being good is at least $\epsilon/2$. This means that the probability of a pair being good in any row corresponding to $A \backslash A_0$ is $< \epsilon/2$. Thus, the number of good pairs in $(A \backslash A_0) \times B$ is $< a \cdot \epsilon/2 \cdot b = \epsilon ab/2$. Since $A_0 \cup (A \backslash A_0) = A$ is a disjoint union, we have that the total number of good entries in $A \times B$ is strictly less than $\epsilon ab/2 + \epsilon ab/2 = \epsilon ab$, a contradiction. Hence, $\#A_0 \geq \epsilon a/2$. ■

3.1.1 Security Proof

Theorem 3.4 (Reductionist Security [78])**.** *If the Schnorr signature scheme succumbs to attack by a probabilistic chosen-message existential forger in the random oracle model, then discrete logs can be found.*

Proof. Assume that the forger algorithm $\mathcal{F}$ has success probability ϵ. That is, given input g^x, $\mathcal{F}$ will make queries to the hash function H and a signing oracle Σ, access a stream of random bits, and output a valid signature on a message that was not part of a signature query with non-negligible probability ϵ.

Suppose we are given an instance $y = g^x$ of the discrete logarithm problem. We run $\mathcal{F}$ with input y. For hash queries, we respond with a uniformly random value for the hash, h. For signing queries, we respond with a pair of random values (h, s). We note that the probability of contradicting ourselves by responding to the forger program's queries with random values is small[1].

Let q_h be a bound on the number of hash queries (m_j, r_j) allowed by the forger. Choose, a priori, a random index $j \leq q_h$. The target index j is the message we need a valid forged signature for, though $\mathcal{F}$ need not give a forgery for this message. By the definition of $\mathcal{F}$, it has non-negligible probability ϵ of producing a valid forgery (for *some* message, nonce pair that it queries h for). Thus, with probability at least ϵ/q_h it will forge a signature for (m_j, r_j) (otherwise the total probability would be less than ϵ, a contradiction).

The motivation for this, at a high level, is the verification equation. If we can somehow induce the forger to produce *two* signatures on (m_j, r_j) such that $h_1 \neq h_2$, we can solve for the private key/discrete logarithm x. That is, the verification condition $g^k = g^s(g^x)^{-h}$ implies that $k \equiv s_1 - h_1 x \equiv s_2 - h_2 x \pmod{q}$, and hence

$$x = (s_1 - s_2)/(h_1 - h_2) \pmod{q}.$$

To this effect, we use two copies of the forger $\mathcal{F}$. As noted by [78], we should think of $\mathcal{F}$ as a computer program (so the word 'copies' is appropriate here). Both forgers are given the

[1]Since (h, s) determines the value of r via $r = g^s(g^x)^{-h}$, rare problems may occur. It is possible that after we responded to a hash query on (M, r) and gave response h, we receive a signing query for M and we respond with (h', s). If the random "r" induced by (h', s) is the same as the r that was part of an earlier hash query, and we respond with $h' \neq h$ in the *signing* query right now, we must restart the procedure. Similarly, if M is queried for a second time to the signing oracle with response (h', s') it is possible that the random r' induced by (h', s') equals r; in that case, we have inconsistent responses (the two H queries for (M, r)). If this happens, we must restart the procedure, since the forger is no longer guaranteed to output a correct answer with probability ϵ. The probability of either occurence is negligible.

public key g^x, the same sequence of random bits, and the same random answers to their queries for hash function values and signatures until they both ask for $H(m_j, r_j)$. We give two independent random answers to *this* hash function query, as illustrated in Figure 3.1. From that point, we give the second copy of the forger a *new* source of randomness, and independent random function values for h: this is the essence of the forking lemma. We then hope that *both* copies of the forger produce signatures corresponding to the pair (m_j, r_j). Because of the "forking" in randomness, the requirement that $h_1 \neq h_2$ should be satisfied with high probability. We now determine the probability of this event.

Let A be the set of possible sequences of random bits and random function values that the forgers take up to where they both ask for $H(m_j, r_j)$, and let B be the set of possible random bits and random function values after that. Note that both sets are finite, since the sequences have length j and $q_h - j$. Let S denote the set of all possible sequences and random function values possible during the procedure. Then $S = A \times B$. Let $J = \{1, ..., q_h\}$. For each $j \in J$ let ϵ_j denote the probability that a sequence $s \in S$ leads to a forgery of the jth message. Then $\sum_{j \in J} \epsilon_j = \epsilon$, since ϵ is the total probability that the forger produces a valid signature.

Now suppose that $\#A = a$ and $\#B = b$, so $\#S = ab$. By our notation, for $\epsilon_j ab$ values of $s \in S$ the forger produces a valid signature for (m_j, r_j). Thus, by the Forking Lemma, there are at least $\epsilon_j a/2$ elements of A that have the following property: the *remaining* part of the forgery algorithm has probability at least $\epsilon_j/2$ of leading to a signature for (m_j, r_j).

For each element of A with this property, the probability that *both* copies of the forger lead to signatures for (m_j, r_j) is at least $(\epsilon_j/2)^2$ (recall that the sequences of random bits and function values at the split become independent after j). In other words,

$$
\begin{aligned}
&\Pr(\mathrm{ver}_K(m_j, \mathcal{F}(\alpha, \beta)) = \text{true} \wedge \mathrm{ver}_K(m_j, \mathcal{F}(\alpha, \gamma)) = \text{true}) \\
&= \Pr(\mathrm{ver}_K(m_j, \mathcal{F}(\alpha, \beta)) = \text{true} \wedge \mathrm{ver}_K(m_j, \mathcal{F}(\alpha, \gamma)) = \text{true} \mid \alpha \in A_0) \cdot \Pr(a \in A_0) \\
&= \Pr(\mathrm{ver}_K(m_j, \mathcal{F}(\alpha, \beta)) = \text{true} \mid a \in A_0) \cdot \Pr(\mathrm{ver}_K(m_j, \mathcal{F}(\alpha, \gamma)) = \text{true} \mid \alpha \in A_0) \cdot \frac{\#A_0}{\#A} \\
&\geq (\epsilon_j/2)(\epsilon_j/2) \left(\frac{\epsilon_j a/2}{a} \right) \\
&= \epsilon_j^3/8.
\end{aligned}
$$

Recalling that j was chosen uniformly at random from J, the probability that the procedure described leads to two valid signatures on some message is at least

$$\frac{1}{q_h} \sum_{j \in J} \frac{\epsilon_j^3}{8} = \frac{1}{8q_h} \sum_{j \in J} \epsilon_j^3.$$

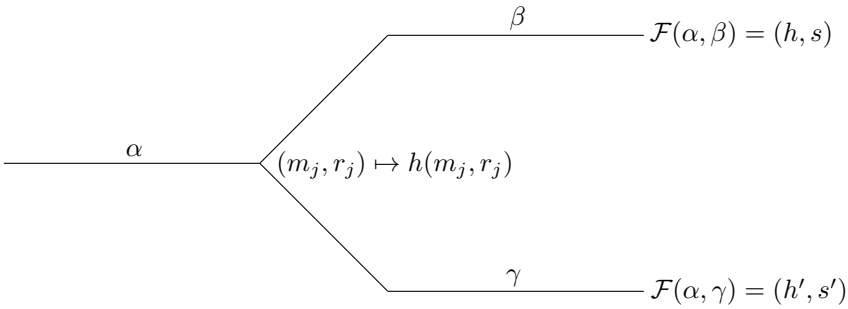

Figure 3.1: Two copies of the forger are supplied with sequence α of bits, random function values until index j

This expression reaches its minimum value when all ϵ_j are equal (i.e. this is a lower bound; the probability can only be greater than this). Since $\sum_{j \in J} \epsilon_j = \epsilon$, we have $\epsilon_j = \epsilon / q_h$. The probability of success in one iteration of the procedure is therefore at least

$$\frac{1}{8q_h} \cdot q_h \left(\frac{\epsilon}{q_h} \right)^3 = \left(\frac{\epsilon}{2q_h} \right)^3 .$$

The probability that we fail to find the private key after k tries is thus $(1 - (\epsilon/2q_h)^3)^k$, which approaches zero as the number of attempts increases. This finishes the proof of our claim. ■.

Tightness and Interpretation

As Koblitz and Menezes point out [77], a signature scheme being existentially unforgeable under adaptive chosen-message attack does not provide defense against all real-world adversaries. There are many practical attacks that fall outside the "provable security" model.

In addition, we note that the proof of Theorem 3.4 is not "tight". This means that under reasonable selections for the parameters ϵ, q_h, the private key must be made larger than is practical for a useful security level. The proof by Pointcheval and Stern [96] for Schnorr signatures is still not tight, but does not lose quite as much efficiency as the above proof. Nevertheless, it is still considered non-tight and does not allow us to choose private keys short enough under conservative ϵ, q_h.

We note that there are constructions in the literature which have tight reductions to computational problems in a group, e.g. Goh and Jarecki [59] (later improved by Katz and Wang [75], for decisional Diffie-Hellman).

Finally, we comment on the random oracle assumption. In the above proof, h is modelled as a random function, meaning that its outputs are sampled uniformly from $\mathbb{Z}_q$ for each query. This property is not attributable to any concrete hash function, since it must be public and *efficiently computable* by anyone. There are schemes which are secure in the random oracle model, but insecure when any concrete hash function is substituted [17]. However, this does not appear to be a problem in practice for the Schnorr signature scheme [78].

3.2 Edwards Curves

To introduce Edwards curves, we consider an example over $\mathbb{Q}$ from [80]. Consider the set of rational points on the unit circle

$$x^2 + y^2 = 1.$$

We can define a group law on these points analagous to multiplication of roots of unity in $\mathbb{C}$; addition is defined by taking the sum of the angles of the inputs. Here, the angles α_i are measured from the positive y-axis (i.e. 12 o'clock), counting clockwise. The addition rule is:

$$\begin{aligned}(x_1, y_1) + (x_2, y_2) &= (\sin\alpha_1, \cos\alpha_1) + (\sin\alpha_2, \cos\alpha_2) \\ &= (\sin(\alpha_1 + \alpha_2),\ \cos(\alpha_1 + \alpha_2)) \\ &= (\sin\alpha_1\cos\alpha_2 + \sin\alpha_2\cos\alpha_1,\ \cos\alpha_1\cos\alpha_2 - \sin\alpha_1\sin\alpha_2) \\ &= (x_1y_2 + x_2y_1,\ y_1y_2 - x_1x_2).\end{aligned}$$

The claim that $\mathbb{T} = (\{(x, y) \in \mathbb{Q}^2 : x^2 + y^2 = 1\}, +)$ is a group with the above composition law is clear. In the context of this circle group, the doubling operation is simply doubling angles, and can be done with the above formulas; there is no need to derive a separate formula for composing a point with itself.

Now, we may compute scalar multiples of the point $\left(\frac{3}{5}, \frac{4}{5}\right) \in \mathbb{T}$: $2\left(\frac{3}{5}, \frac{4}{5}\right) = (x_3, y_3)$, where

$$\begin{aligned}x_3 &= 2 \cdot \frac{3}{5} \cdot \frac{4}{5} = \frac{24}{25}, \\ y_3 &= \frac{4}{5} \cdot \frac{4}{5} - \frac{3}{5} \cdot \frac{3}{5} = \frac{7}{25}.\end{aligned}$$

Note that the denominator of the x-coordinate will always be 5^n for point $nP \in \mathbb{T}$. Thus, the discrete logarithm problem in this group for this choice of base point is easy: we can inspect the power of 5 in the denominator to determine the scalar n. To eliminate this weakness, we can work over a finite field $\mathbb{F}_p$ where $p \equiv 3 \bmod 4$ instead of $\mathbb{Q}$. In this case, $\mathbb{T}$ becomes a subgroup of $\mathbb{F}_{p^2}^*$. To see this, note that $\mathbb{F}_{p^2} \cong \mathbb{F}_p[x]/\langle x^2+1 \rangle$, and that multiplying two elements in this quotient ring is the same as the group law defined above; taking the group of units gives the group correspondence. Another weakness emerges, however; there are subexponential time algorithms for solving the discrete logarithm problem in the multiplicative group of a finite field, e.g. the classic Coppersmith Method [46]. Thus, we still need a different group if it is to be useful in cryptographic computations. There is a way to tweak the definition of $\mathbb{T}$ such that we obtain an elliptic curve group, essentially equivalent to the strongest Weierstrass curves (in terms of discrete log problem difficulty) over prime fields.

Let $\mathbb{K}$ be a field of characteristic greater than 2. An *Edwards curve* over a field $\mathbb{K}$ is a plane curve

$$E : x^2 + y^2 = 1 + dx^2y^2$$

where $d \in \mathbb{K} \backslash \{0, 1\}$. The affine addition law for $(x_1, y_1), (x_2, y_2) \in E(\mathbb{F}_q)$ is given by [50]:

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2} \right).$$

Suppose now that d is not a square in $\mathbb{K}$. The addition law also works for the additive identity $(0, 1)$ and for additive inverses. (For a proof that $P_1 + P_2 \in E(\mathbb{K})$ for all $P_1, P_2 \in E(\mathbb{K})$, see the proof of Theorem 3.1 in [26].) This is not the case in the Weierstrass model, where there are exceptional points for the group law (e.g. point at infinity, additive inverses). The group law is said to be *complete* since it is defined for all pairs of inputs [2].

The proof of completeness for Edwards curves is given next [22, 50].

Theorem 3.5 (Completeness [26]). *Let $P_1 = (x_1, y_1), P_2 = (x_2, y_2) \in E(\mathbb{K})$, where E is an Edwards curve with $d \neq 0, 1$ and d not a square. Then $dx_1x_2y_1y_2 \neq \pm 1$.*

Proof. Assume that $\epsilon = dx_1x_2y_1y_2 = \pm 1$. Then $\epsilon^2 = 1$. Now, $x_1, x_2, y_1, x_2 \neq 0$ since $\mathbb{K}$ is

[2] The Weierstrass formulas can be made complete [34, 32], although there is a small loss of efficiency over the incomplete formulas. By taking advantage of commonalities in the computation of point addition and doubling, one can avoid most of the overhead related to making the group law for Weierstrass curves complete (as in [34]). Longa reports that this overhead is less than 10 percent in practice [82].

a field, and hence an integral domain. Furthermore,

$$\begin{aligned} dx_1^2y_1^2(x_2^2+y_2^2) &= dx_1^2y_1^2(1+dx_2^2y_2^2) \\ &= dx_1^2y_1^2 + d^2x_1^2y_1^2x_2^2y_2^2 \\ &= dx_1^2y_1^2 + \epsilon^2 \\ &= 1 + dx_1^2y_1^2 \\ &= x_1^2 + y_1^2. \end{aligned}$$

Now we construct an expression of the form $a^2 = db^2$ to show that d must be a square, which we excluded in the hypothesis. Let $a = x_1 \pm \epsilon y_1$. Then

$$\begin{aligned} (x_1 \pm \epsilon y_1)^2 &= x_1^2 + y_1^2 \pm 2\epsilon x_1 y_1 \\ &= dx_1^2y_1^2(x_2^2+y_2^2) \pm 2dx_1^2y_1^2x_2y_2 \\ &= dx_1^2y_1^2[x_2^2+y_2^2 \pm 2x_2y_2] \\ &= dx_1^2y_1^2(x_2 \pm y_2)^2. \end{aligned}$$

Now, if $x_2 + y_2 \neq 0$ or $x_2 - y_2 \neq 0$, then d must be a square by the above equation, a contradiction. Otherwise, if $x_2 + y_2 = 0$ and $x_2 - y_2 = 0$, then $x_2 = y_2 = 0$, but (0,0) is not in $E(\mathbb{K})$. ■

Bernstein et al. [22] extended this class of curves to improve efficiency in elliptic curve arithmetic, and exhibited a correspondence with the Montgomery model of elliptic curves. We fix some notation, following that paper. We continue to work over finite fields $\mathbb{K}$ of characteristic greater than 2.

Definition 3.6 (Twisted Edwards Curve [22])**.** A twisted Edwards curve is a curve with distinct parameters $a, d \in \mathbb{K}$, $d \notin \{0, 1\}$ given by the equation

$$E_{E,a,d} : ax^2 + y^2 = 1 + dx^2y^2.$$

An Edwards curve is a twisted Edwards curve with $a = 1$.

Definition 3.7 (Montgomery Curve [22])**.** Let $A \in \mathbb{K}\backslash\{-2, 2\}$ and $B \in \mathbb{K}\backslash\{0\}$. A Montgomery curve with coefficients A and B is given by the equation

$$E_{M,A,B} : Bv^2 = u^3 + Au^2 + u.$$

The Montgomery form has been known for some time, as Montgomery published this elliptic curve parametrization in 1987 [91]. It is notable for its efficient x-coordinate-only scalar multiplication, which is a set of recurrences called the Montgomery Ladder.

In the above definitions the choice of parameters determines many properties of the curve. For instance, we can classify curves up to birational equivalence [108] (for an example with Weierstrass curves, see Chapter 3 of [64]). A birational equivalence is essentially the same as an isomorphism, however, we must remove singularities to obtain a true isomorphism. This is done by homogenizing the curve equation and re-defining the mapping in terms of projective coordinates (also called the projective closure).

In practice these maps allow points on a given curve to be represented as points on an isomorphic curve. This is useful in some instances for computation. Efficient arithmetic in one form can be exploited in some computations, the result converted to another form (e.g. for more efficient point decompression). Bernstein et al. [22] give a birational equivalence between twisted Edwards curves and other forms (as well as enumerating exceptional points for the mapping). We collect a few definitions and facts below. For a careful treatment of algebraic geometry background, standard references include [108, 65, 110].

Definition 3.8 (Morphisms [108])**.** Let X, Y be irreducible algebraic plane curves (e.g. elliptic curves). A morphism ϕ is a rational map from X to Y. A homomorphism is a morphism that respects the group operation on X. A birational equivalence ϕ is a rational map from X to Y that has a rational inverse.

Theorem 3.9. *Morphisms that map the neutral element in X to the neutral element in Y are homomorphisms. Removing singularities makes birational equivalence an isomorphism.*

In the above, we have omitted extension fields. Elliptic curves over finite fields have rational points in the algebraic closure $\overline{\mathbb{K}}$. This is important to note when defining isomorphisms: existence of an isomorphism over the *base field* $\mathbb{K}$ is equivalent to solving a polynomial of some degree over $\mathbb{K}$. Since $\mathbb{K}$ is not algebraically closed, this is not always possible. It is then natural to ask: what is the number of non-isomorphic curves in each model over the base field $\mathbb{K}$? This question has been answered completely by Farashahi et al. [99] for the case of twisted Edwards curves, in agreement with the rough estimates in [22]. René Schoof proved that there are approximately $2p$ isomorphism classes of elliptic curves in total over $\mathbb{F}_p$ [104]. See Table 3.1 for comparison.

Curves isomorphic over finite extensions of $\mathbb{K}$ are also interesting to consider. Two elliptic curves isomorphic over $\overline{\mathbb{K}}$ but not $\mathbb{K}$ are said to be *twists*. Constructing an extension of the base field $\mathbb{K}$ by adjoining roots of an irreducible polynomial gives rise to arithmetic in the extension. We note that the curve orders may be different for curves that are twists of each other. This fact is exploitable in attacks on an elliptic curve that has a twist with smooth group order. This is referred to as twist-insecurity. There is only one quadratic

Table 3.1: Number of isomorphism classes of Edwards curves and twisted Edwards curves [99]

	$p \equiv 1 \bmod 4$	$p \equiv 3 \bmod 4$
Edwards curves	$\approx (2/3)p$	$\approx (3/4)p$
Twisted Edwards curves	$\approx (5/6)p$	$\approx (3/4)p$
All elliptic curves	$\approx 2p$	

twist up to isomorphism, except in the special cases where the j-invariant of the curve is 0 or 1728 [44, 51].

Theorem 3.10 (Equivalence of twisted Edwards and Montgomery [22]). *Let $\mathbb{K}$ be a field with* $\mathrm{char}(\mathbb{K}) \neq 2$.
(i) Every twisted Edwards curve over $\mathbb{K}$ is birationally equivalent over $\mathbb{K}$ to a Montgomery curve.
(ii) Conversely, every Montgomery curve over $\mathbb{K}$ is birationally equivalent over $\mathbb{K}$ to a twisted Edwards curve.

Since twisting involves computations over an extension field, we should prefer to stay in the base field to save time, as long as security is not sacrificed for efficiency. The next result states when an arbitrary elliptic curve can be expressed as an (untwisted) Edwards curve.

Theorem 3.11 (Points of order 4 [22]). *Let $\mathbb{K}$ be a field with* $\mathrm{char}(\mathbb{K}) \neq 2$. *Let E be an elliptic curve over $\mathbb{K}$. The group $E(\mathbb{K})$ has an element of order 4 if and only if E is birationally equivalent over $\mathbb{K}$ to an Edwards curve.*

The next result states when a Montgomery curve is equivalent to an (untwisted) Edwards curve.

Theorem 3.12 (Some Montgomery curves are Edwards curves [22]). *If $\mathbb{K}$ is a finite field with $\#\mathbb{K} \equiv 3 \bmod 4$ then every Montgomery curve over $\mathbb{K}$ is birationally equivalent over $\mathbb{K}$ to an Edwards curve. If $\mathbb{K}$ is a finite field with $\#\mathbb{K} \equiv 1 \bmod 4$ and $E_{M,A,B}$ is a Montgomery curve so that $(A+2)(A-2)$ is a square and δ a nonsquare, then exactly one of $E_{M,A,B}$ and its nontrivial quadratic twist $E_{M,A,\delta B}$ is birationally equivalent to an Edwards curve. In particular, $E_{M,A,A+2}$ is birationally equivalent to an Edwards curve.*

For completeness, we give each birational equivalence in Table 3.2 that is part of the proof of Theorem 3.10. Note that the theorem says that $E_{M,A,B}$ is birationally equivalent

to *some* twisted Edwards curve. This is denoted in the table by specifying the twisted Edwards curve parameters that determine a curve equivalent to $E_{M,A,B}$ over $\mathbb{K}$. The same holds for $E_{E,a,d}$ being birationally equivalent to some Montgomery curve. Other birational equivalences that arise in special cases appear in the proofs of the above theorems, so we omit these and refer to [22].

	$E_{M,A,B} : Av^2 = u^3 + Bu^2 + u$ | $E_{E,a,d} : ax^2 + y^2 = 1 + dx^2y^2$
$(u, v) \to (x, y)$	$(u, v) \mapsto (x, y) = (u/v, (u-1)/(u+1))$ $a = (A+2)/B, d = (A-2)/B$
$(u, v) \leftarrow (x, y)$	$(x, y) \mapsto (u, v) = ((1+y)/(1-y), (1+y)/(1-y)x)$ $A = 2(a+d)/(a-d), B = 4/(a-d)$

Table 3.2: Mappings used in proof of Theorem 3.10

Even with the above results, an elliptic curve might not be equivalent to an Edwards curve over $\mathbb{K}$. In other words, a Weierstrass curve $y^2 = x^3 + ax + b$ may be a member of an isomorphism class that is inequivalent to even a twisted Edwards curve unless it has a point of order 4 (Theorem 3.11). Bernstein et al. have described how to use isogenies to benefit from the twisted Edwards form for computation. This is similar to using the result of Brier and Joye to consider "$a_4 = -3$" Weierstrass curves without much loss of generality. Since we primarily work with twisted Edwards curves, this detail will be omitted.

Via point counting, Bernstein et al. made heuristic arguments for the number of curves birational to a fixed form, of each type of order. This was an improvement over the original coverage of Edwards curves, a useful generalization that covers more curves, and introduces efficiency improvements that we discuss in Section 3.4.

3.3 EdDSA

In this section we consider the features and design of the signature scheme proposed by Bernstein et al. [25]. The scheme is an adaptation of Schnorr's scheme that is suited to batch verification.

EdDSA Parameters

Let $b \geq 10$ be an integer; let $H : \{0,1\} \to \{0,1\}^{2b}$ be a cryptographic hash function; let $q \equiv 1 \bmod 4$ be a prime power. Encode the elements of $\mathbb{F}_q$ with $(b-1)$ bits. Let

$d \in \mathbb{F}_q \backslash \{0,1\}$ be a non-square. Let $2^{b-4} < \ell < 2^{b-3}$ be a prime such that $\ell B = \mathcal{O} = (0,1)$, where $B \neq (0,1)$ is in the group of rational points on a twisted Edwards curve:

$$B \in E(\mathbb{F}_q) := \{(x,y) \in \mathbb{F}_q \times \mathbb{F}_q : -x^2 + y^2 = 1 + dx^2y^2\}. \tag{3.1}$$

We have already shown in Theorem 3.5 that the Edwards addition law is complete for nonsquare d; we note that the theorem holds for twisted Edwards curves as well. On this twist, the addition law is

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 + x_1x_2}{1 - dx_1x_2y_1y_2} \right).$$

Since $q \equiv 1 \bmod 4$, -1 is a square in $\mathbb{F}_q$ (by Euler's theorem). Thus, the twisted Edwards curve is isomorphic to the Edwards curve $E_{E,1,d/i}$ over $\mathbb{F}_q$ via $(x,y) \mapsto (x/i, y)$, where $i = \sqrt{-1} \in \mathbb{F}_q$.

Elements $(x,y) \in E$ are encoded/compressed as b-bit strings $\underline{(x,y)}$. This notation means that we preserve the entire $(b-1)$ bits of y, an element of $\mathbb{F}_q$, and a sign bit which determines x. For this sign bit to be well defined, we need an ordering on field elements. This is accomplished by letting $x \in \mathbb{F}_q$ be negative if the $(b-1)$ bit encoding of x is lexicographically larger than the $(b-1)$ bit encoding of $-x$. For example,

$$x = 3 \in \mathbb{F}_{17}, \quad -x = 14 \in \mathbb{F}_{17}, \quad 00011 > 01110,$$

hence x is negative in this case (when we use the little-endian representation). As will be the case in this thesis, for q an odd prime with little-endian representation of $\mathbb{F}_q$, the negative elements will be $\{1, 3, 5, \ldots, q-2\}$. The x-coordinate can be recovered from $\underline{(x,y)}$ via the curve equation: $x = \pm\sqrt{(y^2-1)/(dy^2+1)}$.

Key Generation
Let k be a random b-bit string, the EdDSA private key. Denoting $H(k) = (h_0, h_1, \ldots, h_{2b-1})$, we compute

$$a = 2^{b-2} + \sum_{3 \leq i \leq b-3} 2^i h_i,$$

which is an integer in the set $\{2^{b-2}, 2^{b-2}+8, \ldots, 2^{b-1}-8\}$. This determines the point $A = aB$ and public key $\underline{A}$.

Signature Generation
To sign a message M with secret key k,

1. compute $r = H(h_b, \ldots, h_{2b-1}, M)$;
2. compute $R = rB$;
3. compute $S = (r + H(\underline{R}, \underline{A}, M)a) \bmod \ell$,

and the signature on M is then $(\underline{R}, S)$ which is $2b$ bits long. When applied to an integer, underlining denotes taking the b-bit little endian encoding.

Signature Verification
To verify a signature $(\underline{R}, S)$ on a message M under public key $\underline{A}$,

1. decompress $\underline{A}$ and $\underline{R}$ to obtain A and R;
2. compute $h = H(\underline{R}, \underline{A}, M)$;
3. verify that $8SB = 8R + 8H(\underline{R}, \underline{A}, M)A$ holds,

and return true if so. If not, the signature is invalid and is rejected. If parsing the strings which purportedly correspond to curve points fails, the signature is also rejected.

Checking correctness of verification is straightforward: in legitimate signing, we have $S = r + ha \bmod \ell$. Taking this scalar multiple of base point B, we have $SB = rB + haB$ in E since the order of B is ℓ. This is equivalent to $R + hA$, which is computed again during verification.

3.3.1 Remarks on Design

Given the description of EdDSA, we now discuss the features that defeat known attacks. We give a side-by-side comparison of EdDSA and Schnorr, in the notation of [25, 105].

In Table 3.3 we make the identification of g with B and g^a with A, the description for Schnorr taken in the additive notation usually employed with elliptic curves.

Verification in Schnorr's scheme using this notation requires computation of $\tilde{R} = SB - H(R, M)A$, based on data from a claimed signature on M. Then we check that $H(\tilde{R}, M) = H(R, M)$ (cf. Section 3.1).

Note that ECDSA hashes only M (cf. equation 2.2). This means that if H ever turned out to have efficiently-computable collisions, a forgery would be immediate. Including the nonce R in the hashing prevents this, since in both Schnorr and EdDSA these differ with

EdDSA	Schnorr
$G = \langle B \rangle$	$G = \langle g \rangle$
$\lvert G \rvert = \ell$, large prime	$\lvert G \rvert = \ell$, large prime
$H : \{0,1\}^* \to \{0,1\}^{2b}$	$H : \{0,1\}^* \to \mathbb{Z}_q$
$G \leq E(\mathbb{F}_q)$	$G \leq \mathbb{F}_p^*$, $\ell \mid p-1$
Private key a, public key $A = aB$	Private key a, public key $A = aB$
$r = H(h_b, \ldots, h_{2b-1}, M)$	$r \in_R \{0, \ldots, \ell - 1\}$
Sign: $R \leftarrow rB$, $S \leftarrow r + H(\underline{R}, \underline{A}, M)a \bmod \ell$	Sign: $R \leftarrow rB$, $S \leftarrow r + H(R, M)a \bmod \ell$
Signature: $(\underline{R}, S)$	Signature: $H(R, M), S$

Table 3.3: Side-by-side comparison of the EdDSA and Schnorr signature schemes

high probability for different messages. EdDSA goes one step further and includes the public key string $\underline{A}$ in hashing. This is done to prevent any attack that computes many hash values; by this description, one must commit to a particular public key.

Also note the way in which the scalar r is generated in each case. EdDSA computes r deterministically, based on half of the private key and the message. This has several nice practical security benefits; for one, we do not need a high-quality random number generator used for each message signed, as needed in ECDSA and Schnorr. If it is possible to guess some bits of per-message private keys due to poor random number generation, the entire key can be recovered using the attack described by Nguyen and Shparlinski [94].

Finally, note that there are no inversions modulo ℓ. This is in contrast to ECDSA, where the signer and verifier both compute inverses. The concern is that the nonce k in ECDSA signing is inverted during the procedure, which can lead to weaknesses when inversion is not implemented in constant time, as k must be kept secret in order to preserve the secret key. A full description of sequential modifications to ElGamal signatures resulting in EdDSA can be found in Daniel Bernstein's blog post [21].

The similarity to Schnorr signatures is clear from the above, with some modifications that do not compromise security. At present, there is no proof that EdDSA is secure in the standard model of signature security (cf. Chapter 1), though it is conjectured to be secure by the authors [3].

3.3.2 Ed25519

Curve25519 is an elliptic curve in which one performs x-coordinate only scalar multiplication and maps the result to $\mathbb{F}_p$. We obviously need an elliptic curve on which to define the

scalar multiplication of a point; the Montgomery curve

$$E : y^2 = x^3 + 486662x^2 + x$$

over $\mathbb{F}_q$ where $q = 2^{255} - 19$ and $A = 486662$ serves as the choice for curve25519. Note that $q \equiv 1 \mod 4$. Bernstein proposed this curve and proved that the Montgomery ladder has no exceptional cases on Curve25519, so $X(2Q)$ and $X(Q+Q')$ are always correctly output in the form x/z for suitable Q, Q' [20]. After the discovery of Edwards form, Bernstein exhibited a birational equivalence between $y^2 = x^3 + 486662x^2 + x$ and the Edwards curve

$$x^2 + y^2 = 1 + (121665/121666)x^2y^2.$$

Ed25519 is the signature scheme EdDSA taken with this curve and parameter choices $b = 256, H =$ SHA-512, ℓ a 253-bit prime from [20], and $d = -121665/121666$. Note that this curve has order 8ℓ. Computing the quantity

$$x = \sqrt{\frac{1 \pm \sqrt{1 \mp d}}{d}} \mod (2^{255} - 19)$$

gives us the x-coordinate of a point of order 8 on E; the point is of the form (x, x). The nontrivial quadratic twist of Curve25519 has order $4\ell'$, for some prime ℓ' of size roughly the same as ℓ. Thus, both the curve and its quadratic twist have curve orders with small cofactor. This is useful in defeating invalid curve attacks on ECDH which use points of small order and Chinese remaindering to attack the public key; designing a curve that has this property, along with its twist allows simpler implementations that do not have to do extra checks on inputs to ensure they are valid.

Recalling the design of EdDSA, public keys are scalar multiples of B, where 8 divides the scalar (as an integer). This means that an attacker attempting to determine the secret key via an active attack (e.g. in ECDH) asks for aP, for P of his choice. If P has small order, it will still satisfy the curve equation and be a valid public key. However, since we can factor a as $8 \cdot a'$, it will always return the identity $(0, 1)$ for all points of order $2, 4, 8$. This defeats the so-called small subgroup attacks; roughly, security is not degraded due to an attacker being able to determine the secret scalar modulo small divisors of the group order. We also note here that the 'choice' of curve order obviously avoids the Pohlig-Hellman key-only attack.

3.4 Coordinate Systems

The classic paper of Cohen, Miyaji and Ono [45] considers several projective coordinate systems, and does a thorough comparison of the costs in each case, taking into account the

inversion-to-multiplication ratio over the finite field $\mathbb{F}_q$. We define projective coordinates for Edwards curves via change of variables and introducing a third coordinate to homogenize the degrees of each term in the curve equation

$$x^2 + y^2 = 1 + dx^2y^2 \leftrightarrow X^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2.$$

We obtain the projective closure of $E_{E,1,d}$ with the correspondence

$$(X : Y : Z) \leftrightarrow (X/Z, Y/Z), Z \neq 0, \quad (X : Y : 0) \leftrightarrow (0, 1) \leftrightarrow \infty.$$

As before, the cost of affine addition is high since one inversion is involved, and each of these requires many modular multiplications.

Rewriting the group law in projective coordinates, we obtain Algorithm 5 for addition on Edwards curves; make the above substitutions and clear denominators to obtain an algorithm, and group common sub-expressions. The algorithm costs $10M + 1S + 7add + 2D$, where D denotes multiplication by one of the curve parameters a, d. Doubling (see Algorithm 6) can be done in $3M + 4S + 1D + 7add$.

Algorithm 5 Addition on $E_{E,a,d}$ [22]

INPUT: $P_1 = (X_1 : Y_1 : Z_1), P_2 = (X_2, Y_2, Z_2)$
OUTPUT: $P_1 + P_2$
$A = Z_1 \cdot Z_2$; $B = A^2$; $C = X_1 \cdot X_2$; $D = Y_1 \cdot Y_2$; $E = d \cdot C \cdot D$;
$F = B - E$; $G = B + E$; $X_3 = A \cdot F \cdot ((X_1 + Y_1) \cdot (X_2 + Y_2) - C - D)$;
$Y_3 = A \cdot G \cdot (D - aC)$; $Z_3 = F \cdot G$.

Algorithm 6 Doubling on $E_{E,a,d}$ [22]

INPUT: $P = (X_1, Y_1, Z_1)$
OUTPUT: $2P$
$B = (X_1 + Y_1)^2$; $C = X_1^2$; $D = Y_1^2$; $E = a \cdot C$; $F = E + D$; $H = Z_1^2$
$J = F - 2H$; $X_3 = (B - C - D) \cdot J$; $Y_3 = F \cdot (E - D)$; $Z_3 = F \cdot J$

Bernstein et al. [22] introduced a new projective coordinate system, called inverted twisted Edwards coordinates. A point (X_1, Y_1, Z_1) on the curve

$$(X^2 + aY^2)Z^2 = X^2Y^2 + dZ^4$$

with $X_1Y_1Z_1 \neq 0$ corresponds to the affine point $(Z_1/X_1, Z_1/Y_1)$ on $E_{E,a,d}$. This explains the term 'inverted coordinates': the twisting factor a is now a coefficient of Y^2 on the

Coordinates	Source	Addition	Doubling
Edwards	[26]	$10M + 1S + (d/a)$	$3M + 4S$
Edwards	[26]	$10M + 1S + (a, a, d)$	$3M + 4S$
Twisted Edwards	[26]	$10M + 1S + (a, d)$	$3M + 4S + (a)$
Inverted Edwards	[27]	$9M + 1S + (d/a)$	$3M + 4S + (d/a)$
Inverted Edwards	[22]	$9M + 1S + (a, a, d)$	$3M + 4S + (a, a, d)$
Inverted twisted Edwards	[22]	$9M + 1S + (a, d)$	$3M + 4S + (a, d)$

Table 3.4: Cost of curve operations in standard and inverted coordinates [22]

left hand side of the projective curve equation, and the affine coordinates are exactly the inverses of the standard projective coordinates. Algorithms for addition and doubling in inverted coordinates are derived similarly. We collect the operation costs from [22] in Table 3.4.

The nice thing about the systems compared in Table 3.4 is the flexibility we gain by having operation cost depend on curve constants, and by using an isomorphic curve. Consider curve25519; the underlying curve can be expressed as

$$E_1 : x^2 + y^2 = 1 + (121665/121666)x^2y^2.$$

This is isomorphic to the twisted Edwards curve

$$E_2 : 121666x^2 + y^2 = 1 + 121665x^2y^2$$

via the mapping $(x, y) \mapsto (x/\sqrt{a}, y)$, where $a = 121666$. Looking at Table 3.4, the difference in cost between addition on E_1 and addition on E_2 is the cost of 1 multiplication by $\bar{d} = 121665/121666$ versus the cost of 1 multiplication by $d = 121665$ and 1 multiplication by $a = 121666$. The bitlength of $\bar{d}$ mod $(2^{255} - 19)$ is approximately 254, giving a clear advantage to operations on E_2; it is a speedup of almost $1M$, though we lose some efficiency in doubling. It is easy to come up with scenarios where this is still a clear win over the whole computation. We also lose some efficiency due to having to compute an isomorphism, i.e. compute $x/\sqrt{a}$. It would help if $\sqrt{a}$ is a small integer, but this is not always necessary; for instance, $\sqrt{-1}$ in $\mathbb{F}_{2^{255}-19}$ is almost full length, and it is still worthwhile to do this. In other words, twisting can save time. Overall, from the table it is clear that inverted coordinates are preferred when many additions must be computed. However, due to the trade-off with multiplication by curve parameters, it is only worthwhile if a, d are both small.

3.4.1 Extended Coordinates

A new proposal for twisted Edwards curve coordinate systems was given by Hisil et al. in [67]. The authors proposed dedicated formulas for addition on twisted Edwards curves, as well as new complete formulas. There is a gain in efficiency by using the dedicated formulas, but implementations using complete formulas are simpler than ones using dedicated formulas. They also noted that multi-core implementations can take advantage of some formulas which turn out to be parallelizable. We record the details for the coordinate system that ends up being used in EdDSA signing/verification [25], namely the extended twisted Edwards coordinates system.

Let us extend affine coordinates (x, y) by adding a product term $t = xy$, so our points look like (x, y, t) on the same twisted Edwards curve $ax^2 + y^2 = 1 + dx^2y^2$ that we have studied. To get projective coordinates, use the mapping

$$(x, y, t) \mapsto (x : y : t : 1).$$

Conversely, a projective point $(X : Y : T : Z)$ corresponds to affine point $(X/Z, Y/Z, T/Z)$ for $Z \neq 0$. The extended coordinate T satisfies $T = XY/Z$. The identity element is $(0 : 1 : 0 : 1)$, and negation is $-(X : Y : T : Z) = (-X : Y : -T : Z)$.

Note that this form is fully compatible with the earlier coordinate systems we have seen; to go from standard projective to extended projective, we must compute $(X : Y : Z) \to (XZ : YZ : XY : Z^2)$. The only other alternative is to just copy X, Y, Z over, and compute $T = XY/Z$, which is clearly inferior to 3 multiplications and one squaring due to the inversion. The converse is trivial: we can drop T from the representation and get a valid standard projective point.

It remains to write down a complete addition formula for these coordinates, and then an addition algorithm that should benefit from the inclusion of T. The addition formula is complete when d is nonsquare and a is square in $\mathbb{K}$:

$$(X_1 : Y_1 : T_1 : Z_1) + (X_2 : Y_2 : T_2 : Z_2) = (X_3 : Y_3 : T_3 : Z_3),$$

where

$$\begin{aligned}
X_3 &= (X_1Y_2 + Y_1X_2)(Z_1Z_2dT_1T_2) \\
Y_3 &= (Y_1Y_2 - aX_1X_2)(Z_1Z_2 + dT_1T_2) \\
T_3 &= (Y_1Y_2 - aX_1X_2)(X_1Y_2 + Y_1X_2) \\
Z_3 &= (Z_1Z_2 - dT_1T_2)(Z_1Z_2 + dT_1T_2).
\end{aligned}$$

Algorithm 7 Addition on $E_{E,a,d}$ in extended projective coordinates [67]

INPUT: $P_1 = (X_1 : Y_1 : T_1 : Z_1), P_2 = (X_2 : Y_2 : T_2 : Z_2)$
OUTPUT: $P_1 + P_2$
$A = X_1 \cdot X_2$; $B = Y_1 \cdot Y_2$; $C = dT_1 \cdot T_2$; $D = Z_1 \cdot Z_2$
$E = (X_1 + Y_1) \cdot (X_2 + Y_2) - A - B$; $F = D - C$; $G = D + C$;
$H = B - aA$; $X_3 = E \cdot F$; $Y_3 = G \cdot H$; $T_3 = E \cdot H$; $Z_3 = F \cdot G$

The addition algorithm follows, and in the most general case costs $9M + 2D$.

As before, using small curve parameters saves time. A mixed addition algorithm results for input in 'affine form', i.e. when $Z_2 = 1$. Further savings are possible, though. Taking Ed25519 as an example, the twisting factor $a = -1$ allows for some simplification, along with the coordinate T. We obtain an $8M + 1D$ addition algorithm, which is effectively $9M$ due to d being treated as a generic element to simplify the implementation.

3.5 Batch Verification

3.5.1 Addition Chains

An addition chain for a given number is a list of numbers that starts with 1, and every number in the list is the sum of two earlier numbers. The given number appears at the end of the sequence. An example of an addition chain is double and add used to compute kP: we start with P and add points in a list according to a fixed rule (without actually needing to store all intermediate results, as the definition of a chain implies). This isn't the best we can do, however: the length of a shortest addition chain for n is proved to be at least $\log_2(n) + \log_2(\mathrm{wt}(n)) - 2.13$ [33]. An easy upper bound on the length of an addition chain that contains n is $\log_2(n) + \mathrm{wt}(n) - 1$, i.e. double and add. It turns out that computing the optimal addition chain is an NP-hard problem, but Bos and Coster proposed some heuristics to obtain a better chain than the simple binary algorithm.

A related notion is that of a vector addition chain. If an addition chain targets a single integer, then a vector addition chain targets a specific vector of integers, and begins with the standard basis vectors e_i, where $e_1 = [1, 0, 0]$, etc. This naturally applies to the multi-scalar multiplication problem: if an optimal chain for $[z_1, z_2, ..., z_N]$ could be computed quickly, batch verification would be a completely solved problem.

DeRooij [47] investigated addition chain performance based in part on the observation

that

$$aP + bQ = a(P + (b \text{ div } a)Q) + (b \bmod a)Q.$$

This observation can be interpreted as a vector addition chain: one recursively applies the above rule until the scalars are 1 and 0, respectively. In practice, the quotient $(b \text{ div } a)$ is almost always 1, except when the batch size N is small. DeRooij notes that a closed formula is difficult to obtain, but on average, the longest vector addition chain obtained by the algorithm he proposes is

$$\tilde{L}(n, N) < \log b / \log(1 + \ln(N)/N),$$

where n is the maximum exponent size of each of the exponents z_i, N is the batch size, and b is the base used to represent exponents. In [47], the requirement $b = \lceil n^{1/N} \rceil$ is set, and thus $\tilde{L}(n, N)$ depends on both n and N.

We note that the inefficiency associated with small batch sizes can be controlled according to DeRooij by changing the base b; there is no problem with the correctness of doing this, as we still only perform elliptic curve additions. The issue is that the chain becomes longer because quotients fail to be 1.

3.5.2 Bos-Coster

EdDSA verifies signatures with a group equation. As with ECDSA*, we may take linear combinations of signature equations and attempt to optimize the computation of the resulting multi-scalar multiplication. In this section, we consider the batching algorithm used by Bernstein et al. in [25] to achieve signing speed records.

Suppose we have message-signature pairs from different signers, denoted $\{(M_1, (R_1, s_1)), ..., (M_N, (R_N, s_N))\}$. A necessary condition for all signatures to be valid is that equation 2.4 is satisfied for all i. Take a random linear combination of verification equations, as in equation 2.4, with scalars $z_i \in [0, 2^\ell]$ where ℓ is the security parameter. The resulting batch verification equation is

$$\sum_{i=1}^{N} z_i(a_i G + b_i Q_i - R_i) = \mathcal{O}.$$

That is,

$$\left(\sum_{i=1}^{N} z_i \cdot a_i \bmod n\right) G + \sum_{i=1}^{N} (z_i \cdot b_i \bmod n) Q_i - \sum_{i=1}^{N} (z_i \bmod n) R_i = \mathcal{O}. \tag{3.2}$$

Note that not all coefficients have full length. Depending on the security parameter ℓ, the coefficients of the R_i may be smaller than n.

Computing the left hand side of verification equation 3.2 is an instance of multi-scalar multiplication. To compute a multi-scalar multiplication, the Bos-Coster method (Algorithm 8) recursively subtracts the largest two scalars, and then performs one point addition. The collection of scalars and elliptic curve points is sorted and updated in each step of the main loop. Note that we must initially maintain storage for all N points A_i and scalars t_i, but as soon as a scalar is zero, we can free the storage associated with the point and the scalar as no more point-addition updates will ever occur for that index. The final result will always come from A_1, and thus when the algorithm terminates, $t_1 = 1$ and $A_1 = z_1P_1 + \cdots + z_NP_N$.

Algorithm 8 The Bos-Coster algorithm

INPUT: $z_1, \ldots, z_N \in_R [0, 2^b - 1]$, $P_1, \ldots, P_N \in E(\mathbb{F}_q)$
OUTPUT: $z_1P_1 + \ldots + z_NP_N$

```
t_i ← z_i for i = i, ..., N and A_i ← P_i for i = 1, ..., N
while t_1 > 0 do
    Sort (t_1, ..., t_N) in descending order, i.e. t_1 ≥ t_2 ≥ ... ≥ t_N, and rearrange the A_i
    accordingly.
    A_2 ← A_1 + A_2
    t_1 ← t_1 − t_2
end while
return A_1
```

Lemma 3.13. Given input $\{(z_i, P_i)\}_{i=1}^N$ where $z_i \in_R [0, 2^b - 1]$, the Bos-Coster algorithm computes

$$z_1P_1 + \cdots + z_NP_N$$

in $\mathcal{O}(N \cdot b/\log_2(N))$ elliptic curve additions.

Below we give justification for this lemma in terms of experimental observations and a heuristic argument, supported by some formal analysis. Our observations for the number of additions required match the asymptotic complexity in [24], which we have recorded as Lemma 3.13.

To estimate the cost of this method at the 128-bit security level, Bos-Coster was tested using the GNU MP library in C for large integer arithmetic (for convenience). The only operations required were subtraction, magnitude comparison, and sorting (generic sorting

N	run 1	run 2	run 3	run 4	run 5	run 6	$64N$	$2^{7.29}N/\log_2(N)$
2	818	1482	2904	522	4550	412	128	313
4	260	261	268	264	282	267	256	313
8	367	363	369	369	368	370	512	417
16	585	584	588	579	576	580	1024	626
32	951	958	967	959	970	974	2048	1002
64	1644	1647	1657	1649	1649	1642	4096	1669
128	2845	2838	2849	2854	2872	2856	8192	2862
256	5006	5031	5047	5028	5020	5035	16384	5008
512	8954	8944	8956	8951	8957	8944	32768	8903
1024	16096	16102	16083	16091	16094	16105	65536	16025
2048	29189	29146	29222	29174	29197	29167	131072	29137
4096	53265	53281	53246	53297	53265	53324	262144	53418

Table 3.5: Bos-Coster running cost in modular subtractions, 128-bit scalars

in C available from stdlib). We also note that the cost reported represents the amount of time it takes to do modular subtractions, and does not include curve operations. Batch sizes were chosen as consecutive powers of two in order to determine the effect on the running time. We report a close fit function by trial and error, namely $2^{7.29}N/\log_2(N)$ which approximates the running time in modular subtractions using the asymptotic formula from Lemma 3.13. By inspecting Table 3.5, we see that the last column converges to the time measurements as the batch size grows. Essentially the same conclusion is drawn when we repeat this test with 256-bit scalars (since the data is quite similar, we do not include it). Therefore, the conclusion of the lemma appears to hold for practical parameter sizes. In particular, the logarithmic speedup claimed in [24] for batch verification is verified.

Here is an informal argument for the asymptotic complexity claimed above. Suppose that we have n sorted nonnegative integers of maximum bit length b. Assuming they are chosen uniformly at random, they are distributed approximately uniformly on the interval $[0, 2^b-1]$. This means the two largest scalars are approximately 2^b-1 and $2^b-1-(2^b-1)/N$. The difference between these two is thus approximately $(2^b-1)/N$. After a subtraction, the largest scalar is replaced with the difference, $\approx 2^b/N$, which is an overall loss of magnitude $\approx \left(\frac{N-1}{N}\right) 2^b$.

We may argue somewhat formally in support of the above intuition. This argument still makes some assumptions, but at least provides some assurance that the above works in practice. Let $X_i, 1 \leq i \leq N$, be uniform independent random variables on $[0, 2^b - 1]$, i.e., random nonnegative integers of length at most b bits. The first step of Algorithm 8

initializes a list of scalars and associated curve points. In the main loop, we sort the scalars in descending order in each iteration, and then subtract the largest two scalars and replace the largest scalar with the result of the subtraction. This step is crucial to our analysis here; we must determine the expected size of this difference.
Let us find the expected magnitude of the difference for the first iteration. Denote by '2max' the function which returns the second-largest of its arguments. We want to compute the expectation

$$\delta := E(\max(X_1, \ldots, X_N) - 2\max(X_1, ..., X_N)). \tag{3.3}$$

Using the definition,

$$\begin{aligned}
\delta &= \sum_{0 \le x_i \le 2^b - 1} \max(x_1, \ldots, x_N) \cdot p(x_1, \ldots, x_N) - \sum_{0 \le x_i \le 2^b - 1} 2\max(x_1, \ldots, x_N) \cdot p(x_1, \ldots, x_N) \\
&= \frac{1}{2^{Nb}} \left(\sum_{0 \le x_i \le 2^b - 1} \max(x_1, \ldots, x_N) - \sum_{0 \le x_i \le 2^b - 1} 2\max(x_1, \ldots, x_N) \right) \\
&= \frac{1}{2^{Nb}} \left(\sum_{i=1}^{2^b - 1} i \cdot i^{N-1} - \sum_{i=1}^{2^b - 1} (2^b - i) i \cdot i^{N-2} \right) \\
&= \frac{1}{2^{Nb}} \left(\sum_{i=1}^{2^b - 1} i^N - \sum_{i=1}^{2^b - 1} (2^b - i) \cdot i^{N-1} \right) \\
&= \frac{1}{2^{Nb}} \left(\sum_{i=1}^{2^b - 1} (2i^N - 2^b i^{N-1}) \right).
\end{aligned}$$

Since N is our batch size, we would like to have an asymptotic formula for the running time in closed form depending on N. To compute this sum, we refer to the literature on Bernoulli polynomials [101, 76]. The sum of the first Nth powers will have the form

$$\sum_{\nu=0}^{m-1} \nu^N = \sum_{k=0}^{N} \binom{N}{k} B_{N-k} \frac{m^{k+1}}{k+1},$$

where B_{N-k} is the $(N-k)$th Bernoulli number. The highest order terms are the only ones that make a difference for large summations, so we re-write this as

$$\sum_{i=0}^{2^b - 1} i^N = \frac{1}{N+1} (2^b - 1)^{N+1} + \frac{1}{2} (2^b - 1)^N + \mathcal{O}\left(\frac{(2^b - 1)^{N-1}}{N+1} \right).$$

Hence, the difference δ can be expressed asymptotically as a function of N:

$$\frac{1}{2^{Nb}}\left(\frac{2\cdot(2^b-1)^{N+1}}{N+1}+(2^b-1)^N-\frac{2^b\cdot(2^b-1)^N}{N}-\frac{2^b\cdot(2^b-1)^{N-1}}{2}+\mathcal{O}\left(\frac{(2^b-1)^{N-1}}{N+1}\right)\right). \tag{3.4}$$

After some simplification, we obtain

$$\delta \approx \frac{2^{b+1}}{N+1}+1-\frac{2^b}{N}-\frac{1}{2} \approx \frac{2^{b+1}}{N+1}-\frac{2^b}{N}. \tag{3.5}$$

As always, one should be careful when interpreting practical significance of asymptotic formulas, since we have made some assumptions about the input behaviour. Nevertheless, this agrees with our earlier informal analysis of the algorithm: the $(b-\log_2(N+1))$-bit result replaces a b-bit scalar. Hence, we have eliminated $\log_2(N+1)$ scalar bits in the first subtraction of the largest two scalars.

What happens for the rest of the algorithm? Does this elimination hold for every subsequent iteration? If the scalar inputs are random integers, we have shown that the *first* step behaves as expected. A linear combination of random variables is still a random variable, so we might expect that the state of the Bos-Coster algorithm will contain data that looks like random integers. However if the attacker has some control over the scalars generated, he can induce denial of service by making the scalars have very large discrepancies. We can also be unlucky in our choice of scalars, but this happens with very low probability. Bernstein et al. note that the algorithm can be modified [25] to handle large discrepancies in the scalars, but it appears too rarely to be worth checking.

Thus, assuming the list behaves as a list of random integers throughout the computation, we eliminate $\log_2(N+1)$ scalar bits in every iteration.

Chapter 4

Pairing-Based Cryptography and Batching

In this chapter we consider the problem of batch signature verification in the context of pairing-based cryptographic protocols. This problem has been studied both theoretically and with concrete implementations of signature schemes [10, 52, 37]. Pairing-based batching appears to be worthwhile as long as the number of invalid signatures is less than 15%, according to Ferrara et al. [52]. Moreover, pairing-based batch forgery identification has been studied as a standalone problem to take advantage of the properties of pairings [81, 84, 85].

Batch signature verification is an important service to provide when efficiency is a priority, for example in Vehicular ad-hoc networks (VANETs) [98]. The number of bits per wireless transmission is particularly crucial, since sending just one bit of data wirelessly is orders of magnitude more expensive in terms of energy consumption than a 32-bit arithmetic operation [14].

VANETs are an active area of research [1, 5]. They are currently standardized in IEEE 1609.2 to use ECDSA with NIST curves [8]. One research direction taken by Zhang et al. [120] explored batch verification of signatures in the context of VANETs, using their custom design signature scheme based on the Camenisch, Hohenberger and Pedersen (CHP) signature scheme for multiple signers [37]. In this chapter, we refer to the pairing counts for Barreto-Naehrig(BN) curves from Rodriguez-Henriquez et al. [103], and apply them to this signature scheme. Finally, we compare the cost of batch verification in ECDSA*/EdDSA using Bos-Coster to the cost of verifying in this context with just three pairing computations and N curve additions.

In Section 4.1, we introduce relevant background to pairings and BN curves. Section 4.2 contains the operation counts for elliptic curve operations and Miller loop computations. Section 4.3 introduces identity-based cryptography and presents the BN-IBV scheme due to Zhang et al. Section 4.4 presents the comparison between BN-IBV and the elliptic curve signature schemes from Chapters 2 and 3.

4.1 Pairings and BN Curves

To formally define a pairing, we need the machinery of divisors on an elliptic curve, as well as some facts about where the n-torsion points of an elliptic curve live. Subject to a simple divisibility condition, we can say that all n-torsion points live in some finite, specified extension of the field of definition via Theorem 4.1.

Theorem 4.1 (Balasubramanian-Koblitz)**.** *Let $E/\mathbb{F}_q$ be an elliptic curve and let $n|\#E(\mathbb{F}_q)$, where n is prime with $\gcd(n,q)=1$ and $n \nmid q-1$. Then*

$$E[n] \subseteq E(\mathbb{F}_{q^k}) \text{ if and only if } n|q^k-1.$$

We call the least positive integer k such that $n|q^k-1$ the embedding degree*.*

The n-torsion group $E[n]$ is isomorphic to $\mathbb{Z}_n \oplus \mathbb{Z}_n$, and so it has $n+1$ subgroups of order n. It can be shown that $E[n] = E(\mathbb{F}_q)[n] \oplus E(\mathbb{F}_{q^k})[n] \cap \ker(\pi_q - [q]))$, where π_q denotes the q-th power Frobenius map. The second factor group also has prime order n, but its components lie in $\mathbb{F}_{q^k}$ and not in $\mathbb{F}_q$. Indeed, the definition of embedding degree k and the condition $k>1$ ensures that $E[n] \not\subseteq E(\mathbb{F}_q)$, and this is needed in order to define a non-degenerate pairing [111].

Let G_1, G_2 be order-n groups such that $E[n] = G_1 \oplus G_2$. Pairings defined on $G_1 \times G_2$ have been classified by the properties satisfied by these two groups as follows . A type 1 pairing is symmetric in the sense that $G_1 = G_2$. A type 2 pairing is asymmetric (i.e. the embedding degree is $k>1$ and $G_1 \neq G_2$) with an efficiently computable isomorphism $\psi : G_2 \to G_1$, which comes from the trace map [31, 39, 111]. Type 3 pairings are constructed from groups where no efficiently computable isomorphism between G_1 and G_2 is known. Security and efficiency of each type of pairing was analyzed and compared in [39], and the authors conclude that Type 3 pairings are at least as secure as Type 2 pairings. Each setting has its own advantages and disadvantages. For instance, for type 2 pairings Vercauteren showed that it is not possible to hash into G_2 [111] in the sense that there is no efficient hash function known for which computing discrete logarithms of hash values is infeasible [39].

4.1.1 Divisors

Let $E : y^2 = x^3 + ax + b$ be an elliptic curve defined over $\mathbb{F}$.

Definition 4.2. A *divisor* D on E is a formal sum of points:

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

where $a_P \in \mathbb{Z}$, and all but finitely many a_P are zero.

The set of divisors on E is denoted by $\mathrm{Div}(E)$, and forms a free $\mathbb{Z}$-module. By "formal sum", we mean that the symbol (P) is simply an identifier for the point P – we might think of these as just polynomials, with no intention of ever evaluating them. Formal sums are unique – there is only one way to write a given divisor. Also note that there are no scalar multiplications or point additions here – a divisor is simply a linear combination of symbols. The *degree* of a divisor is the sum of its coefficients. The set $\mathrm{Div}^0(E)$ denotes the set of divisors of degree 0. We also define a function $\mathrm{Sum} : \mathrm{Div}(E) \to E$, which evaluates the multi-scalar multiplication arising from a given divisor. In other words, $\mathrm{Sum}(D) = \sum_{P \in E} a_P P \in E$, where we omit the brackets to indicate that the elliptic curve group operation is being performed to evaluate the sum. Note that Sum is a surjective homomorphism of groups – it is clearly a homomorphism from $\mathrm{Div}^0(E)$ to E, and since the preimage of $P \in E$ is the divisor $(P) - (\infty)$, it is surjective.

The *function field* $\overline{\mathbb{F}}(E)$ of E is the field of fractions of $\overline{\mathbb{F}}[x, y]/(y^2 - x^3 - ax - b)$. A function $f(x, y) \in \overline{\mathbb{F}}(E)$ is defined at a finite point P if one can write it as a rational function with nonzero denominator at P. Otherwise f is not defined at P and we write $f(P) = \infty$, and say that f has a pole at P. If $f(P) = 0$, f has a zero at P.

Definition 4.3. (order of a point) Let f be a rational function in the function field of E. The muliplicity of a zero or a pole at P is called the *order* of f at P, and is denoted by $\mathrm{ord}_P(f)$.

We omit some details here, namely how to make the order of a point at infinity precise, and the proof that $\mathrm{ord}_P(f)$ is well defined. Proofs can be found in Section 11.1 of [116].

Each element of the function field $\overline{\mathbb{F}}(E)$ determines a *principal divisor*. A principal divisor encodes information about the zeros and poles of f on E. It can be shown that there are only finitely many zeros and poles (assuming f is nonzero in $\overline{\mathbb{F}}(E)$). For a non-zero function f on E, define the divisor of f by

$$(f) = \sum_{P \in E} \mathrm{ord}_P(f)(P) \in \mathrm{Div}^0(E).$$

Lemma 4.4. (properties of divisors [118]) Let E be an elliptic curve and let f, g be non-zero functions on E. Then

1. f has finitely many zeros and poles;
2. $deg((f)) = 0$;
3. f has no zeros or poles if and only if f is a constant;
4. $(f \cdot g) = (f) + (g)$;
5. $(f/g) = (f) - (g)$;
6. $(f) - (g) = 0$ if and only if f is a constant multiple of g.

Definition 4.5. Two divisors D, D' are *equivalent* if $D = D' + (f)$ for some function f. We write $D \sim D'$.

Let $D = \sum_{Q \in E} a_P(Q)$ be a divisor which has disjoint support from f. This means the set of poles and zeros of f is disjoint from the set of points on E for which D has nonzero coefficients. Then we may define the value of f at the divisor D by

$$f(D) = \prod_{Q \in E} (f(Q))^{a_Q}.$$

By the requirement of disjoint supports, $f(D)$ cannot be 0 or ∞. Finally, we note an important result about divisors.

Theorem 4.6 ([116])**.** *Let E be an elliptic curve. Let D be a divisor on E with $deg(D) = 0$. Then there is a function f on E with $(f) = D$ if and only if* $\mathrm{Sum}(D) = \infty$.

As an example, take the divisor $D = n(P) - ([n]P) - (n-1)(\infty)$ where $P \in E[n]$. Then $\mathrm{Sum}(D) = \infty$ since the order of P divides n, and the degree of D is 0 by inspection. Hence, D is the divisor of a rational function over $\overline{\mathbb{F}}$. If $\mathbb{F} = \mathbb{F}_q$ and $P \in E(\mathbb{F}_q)[n]$, then we may assume by Lemma 11.10 of [116] that the rational function is defined over $\mathbb{F}_q$.

4.1.2 The Miller Function

Victor Miller gave an algorithm for evaluating rational functions on divisors [87, 88] as a way to compute the Weil pairing. The same technique is used in computing the Tate pairing.

Definition 4.7 ([115])**.** For every $P \in \mathbb{F}_{q^k}$ and every integer s, let $f_{s,P}$ be an $\mathbb{F}_{q^k}$-rational function with divisor

$$(f_{s,P}) = (s)(P) - ([s]P) - (s-1)(\infty).$$

Such a function $f_{s,P}$ is called a *Miller function*, and is determined uniquely up to multiplication by non-zero elements of $\mathbb{F}_{q^k}$ [115]; a rational function has no poles or zeros changed by multiplication by a constant. We define $f_{0,P} = f_{1,P} = 1$, and $f_{s,\infty} = 1$, since substituting each of these parameters gives an empty divisor.

Theorem 4.8 (Miller Function Properties [88, 115])**.** *Let $a, b \in \mathbb{Z}$, and let $f_{s,P}$ be a Miller function. Then*

1. $f_{a+b,P} = f_{a,P} \cdot f_{b,P} \cdot \frac{\ell_{[a]P,[b]P}}{v_{[a+b]P}}$;
2. $f_{ab,Q} = f_{a,Q}^{b} \cdot f_{b,[a]Q}$;

where $\ell_{[a]P,[b]P}$ is the equation of the line through $[a]P$ and $[b]P$, and $v_{[a+b]P}$ is the equation of the vertical line through $[a+b]P$.

Properties 1 and 2 above are both verified easily by taking divisors of both sides and following through with the definitions. The first property shows that calculating $f_{a,P}(Q)$ is analogous to exponentiation. We can use this in a square-and-multiply algorithm to compute $f_{a,P}(Q)$, given an integer a and points P, Q. Miller's algorithm (Algorithm 9) formalizes this notion and is used in all pairing computations. Property 1 is plainly used in Miller's algorithm to iteratively compute $f_{s,P}$. We will see an application of property 2 when defining the optimal ate pairing in Section 4.1.5.

To see why Algorithm 9 works, note that we start at $\lambda_{\ell-2}$. The first Miller function evaluation is always just 1, since $f_{0,P} = f_{1,P} = 1$. Continuing with double-and-add on the scalar λ, we have:

$$\cdots 2(2(2(\lambda_{\ell-1}) + \lambda_{\ell-2}) + \lambda_{\ell-3}) \cdots .$$

The sequence of computed values after each iteration of the main loop is:

$$f_{2\lambda_{\ell-1}+\lambda_{\ell-2}}, f_{2(2\lambda_{\ell-1}+\lambda_{\ell-2})+\lambda_{\ell-3}}, \cdots$$

Algorithm 9 The Miller loop

INPUT: $P, Q \in E[n]$ and $\lambda = (\lambda_{\ell-1}\lambda_{\ell-2}\ldots\lambda_1\lambda_0) \in \mathbb{N}$
OUTPUT: $f_{\lambda,P}(Q)$

```
T ← P, f ← 1
for i = ℓ − 2 to 0 do
    f ← f² · ℓ_{T,T}(Q) / v_{[2]T}(Q)
    T ← [2]T
    if λ_i ≠ 0 then
        f ← f · ℓ_{T,P}(Q) / v_{T+P}(Q)
        T ← T + P
    end if
end for
return f
```

One obvious modification that reduces the number of nonzero digits in λ is to use a signed digit representation. We can modify Algorithm 9 to take into account such scalars by making the appropriate change to the 'if' statement: if we have a negative scalar bit λ_i, we do $T \leftarrow T - P$ instead, and negate P in the step that updates f. We can also eliminate the vertical line evaluation in the specific case of BN curves, because those denominator values will all be mapped to 1 by the final exponentiation (see Section 4.1.3).

4.1.3 The Tate Pairing

Bilinear pairings are similar to the inner product of linear algebra since they share the following useful properties.

Definition 4.9. Let G_1, G_2, G_T be groups of the same prime order n. A pairing $e_n : G_1 \times G_2 \to G_T$ is bilinear if, for all $P, P_1, P_2 \in G_1$ and $Q, Q_1, Q_2 \in G_2$:

$$e_n(P_1 + P_2, Q) = e_n(P_1, Q)e_n(P_2, Q);$$

$$e_n(P, Q_1 + Q_2) = e_n(P, Q_1)e_n(P, Q_2).$$

A pairing e_n is nondegenerate when both of the following implications hold:

$$\text{if } e_n(P, Q) = 1 \text{ for all } Q \in G_2 \text{ then } P = \infty;$$
$$\text{if } e_n(P, Q) = 1 \text{ for all } P \in G_1 \text{ then } Q = \infty.$$

We give a specialized definition of the Tate pairing in the Type 3 setting for the case where $\#E(\mathbb{F}_q) = n$ is prime. We recall that the embedding degree of E is k.

Theorem 4.10 ([116])**.** *Let* $\mu_n = \{x \in \mathbb{F}_{q^k} | x^n = 1\}$*, let* $G_1 = E(\mathbb{F}_q)$*, let* $G_2 \subset E[n]$ *be the* trace-0 subgroup *[40]. Then there are nondegenerate bilinear pairings*

$$\langle \cdot, \cdot \rangle_n : G_1 \times G_2 \to \mathbb{F}_{q^k}^* / (\mathbb{F}_{q^k}^*)^n$$

and

$$e_n : G_1 \times G_2 \to \mu_n \subset \mathbb{F}_{q^k}^*.$$

The pairing $\langle \cdot, \cdot \rangle_n$ is constructed as follows. Suppose we are given $P \in G_1$ and $Q \in G_2$. Let $D_P \sim (P) - (\infty)$ and $D_Q \sim (Q) - (\infty)$. Let f be a function such that $(f) = nD_P$ (note that f exists by Theorem 4.6, since $\mathrm{Sum}(nD_P) = \infty$ and D_P necessarily has degree 0). Finally, assume that $D_Q = \sum a_i(Q_i)$ and D_P have disjoint support. Then

$$\langle P, Q \rangle_n = f(D_Q) = \prod_i f(Q_i)^{a_i}.$$

Also,

$$e_n(P, Q) = \langle P, Q \rangle_n^{(q^k - 1)/n}.$$

It can be shown that $\langle \cdot, \cdot \rangle_n$ is bilinear and nondegenerate [116].

Computing the Tate pairing

To compute the Tate pairing in general, there is a denominator that has to be computed, along with an expensive exponentiation. Suppose that $P \in E(\mathbb{F}_q)[n]$. Together with the following lemma, we are able to reduce the cost of the pairing significantly because the denominator is always mapped to unity by the final exponentiation.

Lemma 4.11. Let d be a proper divisor of k. Then any nonzero element of $\mathbb{F}_{q^d}$ is mapped to unity by $(\cdot)^{(q^k - 1)/n}$.

Proof. Note that $(q^k - 1) = (q^d - 1) \cdot (q^{k-d} + q^{k-2d} + \cdots + q^d + 1)$. Now, $n \nmid (q^d - 1)$ by definition of the embedding degree. Thus, n must divide the other factor, and since the order of $\mathbb{F}_{q^d}^*$ is $q^d - 1$, exponentiation by $(q^k - 1)/n$ annihalates the subfield exponent. ■

Now, let $P \in G_1$, $Q \in G_2$, $R \in E(\mathbb{F}_q)$ with $R \notin \{\pm P, \pm Q, \infty\}$. Let $D_P = (P+R) - (R) \sim (P) - (\infty)$, $D_Q = (Q) - (\infty)$, and note that the divisors have disjoint support, so our pairing is well-defined. Then there exists a rational function g such that

$$(P+R) - (R) = (P) - (\infty) + (g).$$

Then $n(P+R) - n(R) = n(P) - n(\infty) + (g^n)$, and letting $f = f_{n,P} \cdot g^n$,

$$\begin{aligned} e_n(P,Q) = f(D_Q)^{(q^k-1)/n} &= \left(\frac{f(Q)}{f(\infty)}\right)^{(q^k-1)/n} \\ &= \left(\frac{f_{n,P}(Q) \cdot g^n(Q)}{f_{n,P}(\infty) \cdot g^n(\infty)}\right)^{(q^k-1)/n}. \end{aligned}$$

Note that $g^n(Q), g^n(\infty)$ both get mapped to unity due to the exponentiation by $(q^k-1)/n$. By Lemma 4.11, $f_{n,P}(\infty) \in \mathbb{F}_q$ is also mapped to unity, because it is in a proper subfield. Hence,

$$e_n(P,Q) = f_{n,P}(Q)^{(q^k-1)/n}.$$

4.1.4 The Ate Pairing

In this section, we give the background for the main pairing considered in this thesis, following the derivations in [118]. The ate pairing is a fixed power of the Tate pairing, and is named so because of the shorter Miller loop for this pairing, as we demonstrate in this section.

Let $m \in \mathbb{Z}$. Note that by property 2 of Theorem 4.8,

$$\begin{aligned} e_n^m(P,Q) &= f_{n,Q}(P)^{m(q^k-1)/n} \\ &= \frac{f_{mn,Q}(P)^{(q^k-1)/n}}{f_{m,[n]Q}(P)^{(q^k-1)/n}} \\ &= \frac{f_{mn,Q}(P)^{(q^k-1)/n}}{1^{(q^k-1)/n}}. \end{aligned}$$

The last equality follows because $Q \in E[n]$, and the inductive definition of the Miller function has $f_{m,\infty} = 1$. Thus, we have

$$e_n^m(P,Q) = f_{mn,Q}(P)^{(q^k-1)/n},$$

which defines a non-degenerate pairing whenever n does not divide m. If $n|m$, by definition of μ_n this annihalates the pairing for all P, Q.

Now, we want to find an integer m such that $f_{mn,Q}(P)$ can be written as a multiple or a power of non-degenerate functions $f_{\lambda_i,Q}(P)$ where λ_i is short, as this means fewer Miller loop iterations.

Consider a power of the Tate pairing, $e_n^m(P,Q)$. Let $\lambda \in \mathbb{Z}$ with $\lambda \equiv q \pmod{n}$. Then

$$q^k - 1 \equiv \lambda^k - 1 \pmod{n}.$$

Since $n|q^k - 1$, it follows that $n|\lambda^k - 1$. Consider the exponent $m' = (\lambda^k - 1)/n$. Then by property 2 of Theorem 4.8,

$$\begin{aligned} e_n^{m'}(P,Q) &= f_{nm',Q}(P)^{(q^k-1)/n} \\ &= f_{\lambda^k-1,Q}(P)^{(q^k-1)/n}. \end{aligned}$$

Note again that for non-degeneracy, the exponent m' must not be divisible by n since the reduced Tate pairing produces an n-th root of unity. Now recall property 1 of Theorem 4.8, and apply it to the above to get

$$\begin{aligned} e_n^{m'}(P,Q) &= f_{\lambda^k-1,Q}(P)^{(q^k-1)/n} \\ &= \left(\frac{f_{\lambda^k,Q}(P)}{f_{1,Q}(P) \cdot \frac{\ell_{[\lambda^k-1]Q,Q}(P)}{v_{[\lambda^k]Q}(P)}} \right)^{(q^k-1)/n} \\ &= \left(\frac{f_{\lambda^k,Q}(P)}{1 \cdot \frac{v_{[\lambda^k]Q}(P)}{v_{[\lambda^k]Q}(P)}} \right)^{(q^k-1)/n} \quad \text{since } [\lambda^k - 1]Q = \infty \\ &= f_{\lambda^k,Q}(P)^{(q^k-1)/n}. \end{aligned}$$

We can repeatedly apply property 2 of Theorem 4.8 to obtain

$$\begin{aligned} e_n^{m'}(P,Q) &= f_{\lambda^k,Q}(P)^{(q^k-1)/n} \\ &= \left(f_{\lambda,Q}^{\lambda^{k-1}}(P) \cdot f_{\lambda^{k-1},[\lambda]Q}(P) \right)^{(q^k-1)/n} \\ &= \left(f_{\lambda,Q}^{\lambda^{k-1}}(P) \cdot f_{\lambda,[\lambda]Q}^{\lambda^{k-2}}(P) \cdot f_{\lambda,[\lambda^2]Q}^{\lambda^{k-3}}(P) \cdots f_{\lambda,[\lambda^{k-2}]Q}^{\lambda}(P) \cdot f_{\lambda,[\lambda^{k-1}]Q}(P) \right)^{(q^k-1)/n}. \end{aligned}$$

We can write this compactly as

$$\left(\prod_{i=0}^{k-1} f_{\lambda,[\lambda^{k-1-i}]Q}^{\lambda^i}(P)\right)^{(q^k-1)/n}.$$

Since $\lambda \equiv q \bmod n$, we have

$$e_n^{m'}(P,Q) = \left(\prod_{i=0}^{k-1} f_{\lambda,[q^{k-1-i}]Q}^{q^i}(P)\right)^{(q^k-1)/n}.$$

Noting that for $a \in \mathbb{Z}$,

$$(f_{a,Q}(P))^q = f_{a,\pi_q(Q)}(\pi_q(P)) = f_{a,[q]Q}(P),$$

since $P \in G_1$ gives $\pi_q(P) = P$ and $Q \in G_2$ gives $\pi_q(Q) = [q]Q$. Thus, for $i \geq 0$,

$$f_{a,[q^i]Q}(P) = f_{a,Q}^{q^i}(P).$$

Following through with the above,

$$\begin{aligned} e_n^{m'}(P,Q) &= \left(\prod_{i=0}^{k-1} f_{\lambda,Q}^{q^{i+k-1-i}}(P)\right)^{(q^k-1)/n} \\ &= \left(\prod_{i=0}^{k-1} f_{\lambda,Q}^{q^{k-1}}(P)\right)^{(q^k-1)/n} \\ &= (f_{\lambda,Q}^{k\cdot q^{k-1}}(P))^{(q^k-1)/n} \\ &= (f_{\lambda,Q}(P))^{k\cdot q^{k-1}\cdot(q^k-1)/n}. \end{aligned}$$

Given this result, we want to eliminate the extra exponentiations, while still keeping control over the smaller scalar λ. To achieve this, let $m = m' \cdot (k \cdot q^{k-1})^{-1} \bmod n$ (note that the inverse exists because k and q are relatively prime to n). Then define $a(P,Q) = e_n^m(P,Q)$, which gives

$$a(P,Q) = (f_{\lambda,Q}(P))^{(q^k-1)/n}.$$

We should check that this exponent gives a nondegenerate pairing, i.e., is it true that $n \nmid m$? If we chose λ such that $n^2 \nmid \lambda^k - 1$, then $n \nmid m$ and $n \nmid m'$.

As an example, we could satisfy the above conditions in practice by letting $\lambda = t - 1$, where t is the trace of Frobenius. Then since $\#E(\mathbb{F}_q) = q + 1 - t$ and n is the prime order of a subgroup (so $n|q+1-t$), Hasse's Theorem says that $|t| \leq 2\sqrt{q}$. Thus the length of the Miller loop is approximately halved when making this choice for λ.

BN Curves

When considering elliptic curves for use with the Tate pairing, we must ensure that the known attacks will not prevail, assuming parameter sizes are large enough. In particular, the Frey-Rück attack [55] uses the Tate pairing to map the DLP in G_1 to the discrete logarithm problem in $\mathbb{F}_{q^k}$. Barreto, Lynn and Scott described a method of finding elliptic curves with embedding degrees greater than 6 [15] over large characteristic finite fields. Barreto and Naehrig introduced a family of pairing-friendly curves in [16] that answered a previously open question about generating elliptic curves of prime order with embedding degree k greater than 6.

In the context of pairing groups, a common measure of relative group size is given by $\rho = \log(p)/\log(r)$, where $r|\#E(\mathbb{F}_p)$ is the prime order of a subgroup generator P, and p is the chosen prime for the base field. The work of [16] gave an algorithm to generate prime order curves with $\rho \approx 1$, while previously known methods produced $\rho \geq 1.25$. The ρ value indicates a trade-off in security and efficiency: if p is too large relative to r, field elements (and therefore elliptic curve group elements) are no longer short. If p is too small relative to r, we sacrifice security for efficiency since the embedding degree k is considered fixed.

BN curves are of the form $E/\mathbb{F}_p : y^2 = x^3 + b$ and are parametrized as follows:

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

Here, n and r are the same because the curve generated is of prime order. For details on the choice of these parametrizations, as well as the generation algorithm for BN curves, see [15, 16].

We are interested in the 128-bit security level, so the generation algorithm will output parameters $p(z), n(z), b, y$ where b is the curve constant in short Weierstrass form, and y is part of the subgroup generator $P = (1, y)$ such that the bit-length of p and n are roughly equal to 256. Since $k = 12$, pairing values live in a 3072-bit finite field $\mathbb{F}_{p^k}$ and $G_1 = \langle P \rangle$ has group elements of size 512 bits. Again, recall that for BN curves $\rho \approx 1$, so the size of p is equal to the size of the subgroup order, and hence r is 256 bits, providing 128-bit DLP security.

Optimal Ate Pairing

The *optimal ate pairing* is also a fixed power of the Tate pairing. Vercauteren [115] introduced the notion of an optimal pairing, and gave a method for computing the appropriate

exponent. A pairing is *optimal* if it can be computed in $\log_2(n)/\varphi(k)+\epsilon(k)$ Miller loop iterations, where $\epsilon(k) \leq \log_2(k)$ and φ is the Euler totient function. The ate pairing derived in the last section does not meet this definition, but the following improved pairing does due to the choice of exponent. Details about deriving the exponent for this pairing can be found in [115, 118].

Definition 4.12 ([13])**.** The optimal ate pairing is defined over a BN curve by

$$\begin{aligned} a_{opt} : G_1 \times G_2 &\rightarrow G_T \\ (P,Q) &\rightarrow (f_{\theta,Q}(P) \cdot \ell_{[\theta]Q,\pi_p(Q)}(P) \cdot \ell_{[\theta]Q+\pi_p(Q),-\pi_p^2(Q)}(P))^{\frac{p^{12}-1}{n}}, \end{aligned}$$

where $\theta = 6z+2 \in \mathbb{Z}$, the map $\pi_p : E \rightarrow E$ is the Frobenius endomorphism $\pi_p(x,y) = (x^p, y^p)$; groups G_1, G_2 are determined by the eigenspaces of π_p as $G_1 = E[n] \cap \mathrm{Ker}(\pi_p - [1]) = E(\mathbb{F}_p)[n]$ and $G_2 = E[n] \cap \mathrm{Ker}(\pi_p - [p]) \subseteq E(\mathbb{F}_{p^{12}})[n]$.

Definition 4.13 (BN curve facts)**.** Let E be a BN curve over $\mathbb{F}_p$. Then there is a degree 6 twist, $E'/\mathbb{F}_{p^2}$ such that:

1. $n | \#E'(\mathbb{F}_{p^2})$;
2. The defining equation of E' is $y^2 = x^3 + b/\xi$, where $\xi \in \mathbb{F}_{p^2} \backslash ((\mathbb{F}_{p^2})^2 \cup (\mathbb{F}_{p^2})^3)$;
3. The polynomial $x^6 - \xi \in \mathbb{F}_{p^2}[x]$ is irreducible over $\mathbb{F}_{p^2}$, when $p \equiv 1 \bmod 6$;
4. The twist isomorphism $\psi : E' \rightarrow E$ is given by
$$(x', y') \mapsto (\xi^{1/3}x', \xi^{1/2}y').$$

The sextic twist $E'(\mathbb{F}_{p^2})$ is used for all non-pairing operations, since curve operations will then only require arithmetic over $\mathbb{F}_{p^2}$. One takes the image under the twisting isomorphism $\psi : E'(\mathbb{F}_{p^2}) \rightarrow G_2$ when doing pairing computation.

We note that the coordinates of the image $\psi(Q)$ are in proper subfields of $\mathbb{F}^*_{p^{12}}$. To see this, we write down the tower field extensions explicitly and do a direct computation. We can take two points of view for building the tower extensions:

$$\mathbb{F}_{p^2} - \mathbb{F}_{p^6} - \mathbb{F}_{p^{12}},$$

or

$$\mathbb{F}_{p^2} - \mathbb{F}_{p^4} - \mathbb{F}_{p^{12}}.$$

The two are equivalent, up to permutation of $\mathbb{F}_{p^2}$ coordinates. We have:

- $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2 - \beta)$, where $\beta = -1$
- $\mathbb{F}_{p^4} = \mathbb{F}_{p^2}[s]/(s^2 - \xi)$, where $\xi = i + 1$
- $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi)$, where $\xi = i + 1$
- $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[w]/(w^2 - v)$; or $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^4}[w]/(w^3 - s)$

These extensions are well defined, following from the fact that ξ is neither a square nor a cube in $\mathbb{F}_{p^2}$ (so the corresponding quadratic and cubic polynomials are irreducible over $\mathbb{F}_{p^2}$). Now, let $w \in \mathbb{F}_{p^{12}}$ be a primitive root of the irredudicble polynomial $x^6 - \xi$, so $w^6 = \xi$ in $\mathbb{F}_{p^{12}}$. Then we can rewrite the twisting isomorphism as

$$\psi(x', y') = (w^2 x', w^3 y').$$

By our tower construction, $w^2 = v \in \mathbb{F}_{p^6}$ and $w^3 = s \in \mathbb{F}_{p^4}$. The G_2' coordinates x' and y' are both in $\mathbb{F}_{p^2}$, so $x = w^2 x' \in \mathbb{F}_{p^6}$ and $y = w^3 y' \in \mathbb{F}_{p^4}$. Using this observation, we can eliminate denominator computations from the Miller loop by Lemma 4.11.

For completeness, we present Algorithm 10 from [103] for computing the optimal ate pairing.

4.2 Counting Field Operations

In this section, we give a count for the field operations involved in computing an optimal Ate pairing over a particular BN curve, as in Sanchez et al. [103]. We focus primarily on a first-order approximation of costs of field operations.

Symbol	Defines the cost of:
(a, m, s, i)	addition, multiplication squaring, inversion cost over $\mathbb{F}_p$
$(\tilde{a}, \tilde{m}, \tilde{s}, \tilde{i})$	addition, multiplication squaring, inversion cost over $\mathbb{F}_{p^2}$

Table 4.1: Definitions for symbols counting the cost of field operations

Table 4.1 defines the symbols for field operation costs that we use to parametrize our analysis. The field $\mathbb{F}_{p^{12}}$ is constructed with a sequence of tower extensions of degree 2 and 3, which are shown to have reduction-friendly algorithms for arithmetic in Theorem 1 of [13], assuming that the operands grow up to double precision in a specified way.

Algorithm 10 The Optimal Ate Pairing [103]

INPUT: $P \in G_1$, $Q \in G_2$
OUTPUT: $f = a_{opt}(P, Q)$

$f \leftarrow 1, T \leftarrow Q, s \leftarrow |6u + 2|$
Write $s = \sum_{i=0}^{m-1} s_i 2^i$ with $s_i \in \{-1, 0, 1\}$
for $i = m - 2$ to 0 **do**
    $f \leftarrow f^2 \cdot \ell_{T,T}(P)$, $T \leftarrow [2]T$
    if $s_i = 1$ **then**
        $f \leftarrow f \cdot \ell_{T,Q}(P)$, $T \leftarrow T + Q$
    else if $s_i = -1$ **then**
        $f \leftarrow f \cdot \ell_{T,-Q}(P)$, $T \leftarrow T - Q$
    end if
end for
$f \leftarrow f^{p^6}$
$R \leftarrow \pi(Q)$; $f \leftarrow f \cdot \ell_{-T,R}(P)$; $T \leftarrow T + R$
$R \leftarrow \pi^2(Q)$; $f \leftarrow f \cdot \ell_{-T,-R}(P)$; $T \leftarrow T - R$
$f \leftarrow f^{(p^{12}-1)/n}$
return f

Since the prime p for BN curves is not of a special form suitable for fast reduction, Montgomery arithmetic and reduction is used. To accurately account for the costs, we will have to track more operations than simply base field multiplications. Moreover, 'lazy reduction' can be used to trade off some reductions for extra double-precision operations for performance improvement [13], which defers expensive reductions to a higher layer in the tower of extension fields, and once the operands have grown to double-precision (meaning twice the size of a $\mathbb{F}_p$ element). Base field arithmetic in the implementation from [103] uses modified versions of the Montgomery product. This is the fastest option for primes without naturally reduction-friendly form, and performs well when many field operations are performed before converting back from Montgomery form (as happens routinely in pairing-based cryptography).

$\mathbb{F}_{p^2}$ Arithmetic

Let $a = a_0 + a_1 i$ and $b = b_0 + b_1 i \in \mathbb{F}_{p^2}$. Then $a \cdot b$ is computed using Karatsuba's method which reduces the cost of multiplication from $4m$ to $3m$ over the naive method, at the

expense of some extra additions:

$$\begin{aligned}(a_0 + a_1 i) \cdot (b_0 + b_1 i) &= a_0 b_0 - a_1 b_1 + (a_0 b_1 + a_1 b_0) i \\ &= (a_0 b_0 - a_1 b_1) + ((a_0 + b_0) \cdot (a_1 + b_1) - a_0 b_0 - a_1 b_1) i.\end{aligned}$$

Squaring is done using the complex multiplication method, using just 2 small-field multiplications [103, 39]:

$$\begin{aligned}(a_0 + a_1 i) \cdot (a_0 + a_1 i) &= a_0^2 - a_1^2 + (2 a_0 a_1) i \\ &= (a_0 - a_1)(a_0 + a_1) + (2 a_0 a_1) i.\end{aligned}$$

To compute an inversion, note first that $(a - bi)(a + bi) = a^2 + b^2$. Thus,

$$\begin{aligned}(a + bi)^{-1} &= \frac{a - bi}{a^2 + b^2} \\ &= (a^2 + b^2)^{-1} \cdot (a - bi).\end{aligned}$$

This gives $\tilde{m} \approx 3m$, $\tilde{s} \approx 2m$, $\tilde{i} = 2s + 1i + 2m$. This is a rough approximation, because we have not said anything about how reduction should be done, and we have completely discounted the cost of additions. Since the cost of multiplication is approximately an order of magnitude higher than the cost of addition, this seems to be justified. Moreover, since we aim to compare batching operation counts in BN-IBV to that of EdDSA and ECDSA*, which was also done with rough approximation, we follow in this line, noting where we have simplified the detailed accounting of [103, 13].

$\mathbb{F}_{p^6}$ Arithmetic

To determine the cost of multiplication over $\mathbb{F}_{p^6}$, let $a = a_0 + a_1 v + a_2 v^2$ and $b = b_0 + b_1 v + b_2 v^2 \in \mathbb{F}_{p^6}$. Then

$$\begin{aligned}(a_0 + a_1 v + a_2 v^2) \cdot (b_0 + b_1 v + b_2 v^2) = a_0 b_0 + (a_0 b_1 + a_1 b_0) v + &\quad (a_0 b_2 + a_1 b_1 + a_2 b_0) v^2 \\ + &\quad (a_1 b_2 + a_2 b_1) v^3 + (a_3 b_3) v^4.\end{aligned}$$

Noting the field definitions in Section 4.1, we have $v^3 = \xi$. Reducing the above using this relation, we have

$$\begin{aligned}(a_0 + a_1 v + a_2 v^2) \cdot (b_0 + b_1 v + b_2 v^2) &= (a_0 b_0 + (\underline{a_1 b_2 + a_2 b_1}) \xi) \\ &+ (a_2 b_2 \xi + (\underline{a_0 b_1 + a_1 b_0})) v \\ &+ ((\underline{a_0 b_2 + a_2 b_0}) + a_1 b_1) v^2.\end{aligned}$$

Now, apply Karatsuba's method to each cross-term product (underlined) in the above. This gives an approximate cost of $3\tilde{m} + 3\tilde{m} = 6\tilde{m}$, after discounting the cost of all additions.

An asymmetric squaring formula for cubic extensions was given by Chung and Hasan in [43]:

$$\begin{aligned}(a + bv + cv^2)^2 &= a^2 + b^2v^2 + c^2v^4 + 2abv + 2acv^2 + 2bcv^3 \\ &= (a^2 + 2bc\xi) + (2ab + c^2\xi)v + (\underline{b^2 + 2ac})v^2.\end{aligned}$$

The novel part about the squaring formula is that it makes use of intermediate results from the v^0 and v^1-coefficients to compute the underlined part above:

$$(b^2 + 2ac) = ((a - b + c)^2 - a^2 - c^2 + 2ab + 2bc),$$

which can be verified directly. After discounting addition and reductions/carry checking, this outperforms the naive method (i.e. if we did not know how to compute $b^2 + 2ac$ any way but directly) by $1\tilde{m}$, since we still have to compute the squaring $(a - b + c)^2$ over $\mathbb{F}_{p^2}$. Thus the cost of $\mathbb{F}_{p^6}$ squaring is estimated as $2\tilde{m} + 3\tilde{s}$.

$\mathbb{F}_{p^{12}}$ Arithmetic

Given $a = (a_0 + a_1w)$ and $b = (b_0 + b_1w) \in \mathbb{F}_{p^{12}}$, we compute $a \cdot b$ using Karatsuba's method, resulting in cost $3\times$(cost of multiplications in $\mathbb{F}_{p^6}$), i.e. $18\tilde{m}$.

Given $a = (a + bw)$, compute $a^2 = (a^2 + vb^2) + (2ab)v$ using the same method as for $\mathbb{F}_{p^2}$ since we have a quadratic extension. The cost is estimated as $2\times$(cost of multiplication in $\mathbb{F}_{p^6}$), i.e. $2 \cdot (6\tilde{m}) = 12\tilde{m}$.

Let $G_{\Phi_6(p)}(\mathbb{F}_{p^2}) = \{x \in \mathbb{F}_{p^{12}} : x^{\Phi_6(p)} = 1\} \subset \mathbb{F}^*_{p^{12}}$ be the cyclotomic subgroup corresponding to the 6th cyclotomic polynomial. There is a significant speed-up available for squaring in $G_{\Phi_6(p)}(\mathbb{F}_{p^2})$, due to additional identities satisfied by these elements, as well as free inversions in this subgroup. Recall that

$$(p^{12} - 1) = (p^6 - 1)(p^2 + 1) \cdot \Phi_6(p) = (p^6 - 1)(p^2 + 1)(p^4 - p^2 + 1),$$

so membership in $G_{\Phi_6(p)}(\mathbb{F}_{p^2})$ implies that $x^{p^6+1} = 1$. Rearranging this, we get $x^{p^6} = x^{-1}$, so inversion in this subgroup just costs us a conjugation.

Lemma 4.14 ([113])**.** If $x \in \mathbb{F}_{p^{12}}$ satisfies $x^{p^6+1} = 1$, then x^2 can be computed in 2 squarings over $\mathbb{F}_{p^6}$.

Proof. Let $\alpha = a + bw$, where $a, b \in \mathbb{F}_{p^6}$ satisfy $\alpha^{p^6+1} = 1$. Note that $\alpha^{p^6} = a - bw$. Then

$$1 = \alpha^{p^6+1} = (a + bw) \cdot (a - bw) = a^2 - b^2 w^2 = a^2 - b^2 v.$$

This gives us a relation for a^2 in terms of b^2. Computing directly and using this relation gives the result:

$$\begin{aligned} \alpha^2 = (a + bw)^2 &= a^2 + b^2 w^2 + (2ab)w \\ &= (2b^2 v + 1) + ((a + b)^2 - a^2 - b^2) \\ &= (2b^2 v + 1) + ((a + b)^2 - (b^2 v + 1) - b^2). \end{aligned}$$

Therefore, after discounting the cost of addition, carries/reductions and shifts, we require just 2 squarings over $\mathbb{F}_{p^6}$, namely b^2 and $(a + b)^2$. ■

4.2.1 The Miller Loop

In this section, we examine the cost of a single optimal Ate pairing with Miller loop length $s = 6z + 2$, where $z = -(2^{62} + 2^{55} + 1)$ is selected to be the BN parameter [103]. Recall that the BN curve $y^2 = x^3 + b$ has a degree 6 twist $E' : y^2 = x^3 + b/\xi$. On G'_2, we choose standard projective coordinates, as in [103]. We must strongly separate our notions of arithmetic for computing the bilinear pairing and "ordinary" curve operations; here we *do not* use Jacobian coordinates, as they become more efficient only when doing exclusively group operations. The formulas for computing $2T = (X_3 : Y_3 : Z_3)$ used there are given by [13]:

$$X_3 = \frac{X_1 Y_1}{2}(Y_1^2 - 9b' Z_1^2), \quad Y_3 = \left[\frac{1}{2}(Y_1^2 + 9b' Z_1^2)\right]^2 - 27b'^2 Z_1^4, \quad Z_3 = 2Y1^3 Z_1,$$

where $b = 1 + i$ and $b' = 2/(1 + i) = 1 - i$ (recalling Definition 4.13). Determining the projective formulas for doubling is the same as in Section 3.2 of [64]. For completeness, we include the sequence of operations for computing $2T$ from [13].

$$A = X_1 \cdot Y_1/2,\ B = Y_1^2,\ C = Z_1^2,\ D = 3C,\ E_0 = D_0 + D_1,$$

$$E_1 = D_1 - D_0,\ F = 3E,\ X_3 = A \cdot (B - F),\ G = (B + F)/2,$$
$$Y_3 = G^2 - 3E^2,\ H = (Y_1 + Z_1)^2 - (B + C),$$
$$Z_3 = B \cdot H,\ I = E - B, J = X_1^2$$

After ignoring additions and multiplications by small constants, this sequence of operations requires 3 multiplications and 6 squarings in $\mathbb{F}_{p^2}$.

Now, every time a point doubling is computed in the Miller loop for a_{opt}, there is a tangent line computation. We estimate the cost of this operation here. The affine equation of the tangent line through $T = (x_T, y_T)$, evaluated at $P = (x, y)$ is given by

$$\ell_{T,T}(P) = y - y_T - \left(\frac{3x_T^2}{2y_T}\right)(x - x_T).$$

Note that point doubling is done on the twist, using $\mathbb{F}_{p^2}$-arithmetic only. For tangent line evaluation, arithmetic is over $\mathbb{F}_{p^{12}}$ since the coordinates x_Q, y_Q are in that field and their values are used in arithmetic, as opposed to simple embeddings in a larger/smaller field via isomorphism.

Let $(x_T, y_T) = (u_T w^2, v_T w^3)$ and let (X, Y, Z) be a projective point such that $u_T = X/Z$ and $v_T = Y/Z$ (note that $x_T, y_T \in \mathbb{F}_{p^{12}}$ and $u_T, v_T \in \mathbb{F}_{p^2}$). Then (we omit subscripts for clarity)

$$\begin{aligned}
\ell_{T,T}(P) &= y_P - v_T w^3 - \left(\frac{3(u_T w^2)^2}{2v_T w^3}\right)(x_P - u_T w^2) \\
&= y_P - \frac{Y}{Z}w^3 - \left(\frac{3\frac{X^2}{Z^2}w^4}{2\frac{Y}{Z}w^3}\right)(x_P - \frac{X}{Z}w^2) \\
&= [Z y_P - Y w^3 - \left(\frac{3X^2}{2Y}\cdot w\right)(x_P - \frac{X}{Z}w^2)]/Z \\
&= \frac{Z y_P - Y w^3}{Z} - \frac{3X^2 w^4}{Z^2}\cdot\frac{Z}{2Y w^3}\cdot\frac{x_P Z - X w^2}{Z} \\
&= \frac{Z y_P - Y w^3}{Z} - \frac{3X^2 w(x_P Z - X w^2)}{2Y Z^2} \\
&= \frac{2Y Z^2 y_P - 2Y^2 Z w^3 - 3X^2 Z x_P w + 3X^3 w^3}{2Y Z^2} \\
&= \frac{2Y Z y_P - 2Y^2 w^3 - 3X^2 x_P w + 3(Y^2 - b'Z^2)w^3}{2Y Z} \\
&= \frac{2Y Z y_P - 3X^2 x_P w + (Y^2 - 3b'Z^2)w^3}{2Y Z}.
\end{aligned}$$

Since the denominator is in $\mathbb{F}_{p^2}$, it is mapped to 1 by the final exponentiation. The formula for the line evaluation is thus given by

$$\ell_{T,T}(P) = 2Y_1 Z_1 y_P - 3X_1^2 x_P w + (Y_1^2 - 3b'Z_1^2)w^3.$$

Noting that the sequence of operations for doubling costs $\approx 3\tilde{m} + 6\tilde{s}$, $\ell_{T,T}(P)$ can be computed in just a few additional operations since $X_1^2, Y_1^2, Y_1 \cdot Z_1$ are reused from the doubling step. Multiplication by small constants is implemented with additions (and thus ignored in our counts). Therefore, we only need 4 additional $\mathbb{F}_p$ multiplications when computing $(2Y_1Z_1) \cdot y_P$ and $(3X_1^2) \cdot x_P$. The total cost is approximately $3\tilde{m} + 6\tilde{s} + 4m$ for doubling and line evaluation together (line 4 of Algorithm 10).

Line 6 of Algorithm 10 requires a point addition in G_2' and a secant line evaluation. We can estimate the cost of these computations in the same way. Let $T = (X_1, Y_1, Z_1)$ $Q = (X_2, Y_2, 1) \in E'(\mathbb{F}_{p^2})$ and $P = (x_P, y_P) \in E(\mathbb{F}_p)$. Note that Q is given in affine coordinates since it is the input to the pairing computation, while T is an accumulator with general projective coordinates. The formulas for $(X_3, Y_3, Z_3) = T + Q$ are given by:

$$\begin{aligned} X_3 &= \lambda(\lambda^2 + Z_1\theta^2 - 2X_1\lambda^2) \\ Y_3 &= \theta(3X_1\lambda^2 - \lambda^3 - Z_1\theta^2) - Y_1\lambda^3 \\ Z_3 &= Z_1\lambda^3 \end{aligned}$$

where $\theta = Y_1 - Y_2Z_1$ and $\lambda = X_1 - X_2Z_1$. Computing X_3, Y_3, Z_3 thus requires approximately $9\tilde{m} + 2\tilde{s}$, using the following sequence of operations [13]:

$$t_1 \leftarrow Z_1 \cdot X_2, t_2 \leftarrow Z_1 \cdot Y_2 \tag{4.1}$$
$$t_1 \leftarrow X_1 - t_1, t_2 \leftarrow Y_1 - t_2 \tag{4.2}$$
$$t_3 \leftarrow t_1 \cdot t_1 \tag{4.3}$$
$$X_3 \leftarrow t_3 \cdot X_1, t_4 \leftarrow t_4 \cdot Z_1 \tag{4.4}$$
$$t_4 \leftarrow t_3 + t_4 \tag{4.5}$$
$$t_4 \leftarrow t_4 - X_3 \tag{4.6}$$
$$X_3 \leftarrow X_3 - t_4 \tag{4.7}$$
$$T_1 \leftarrow t_2 \cdot X_3, T_2 \leftarrow t_3 \cdot Y_1 \tag{4.8}$$
$$T_2 \leftarrow T_1 - T_2 \tag{4.9}$$
$$Y_3 \leftarrow T_2, X_3 \leftarrow t_1 \cdot t_4, Z_3 \leftarrow t_3 \cdot Z_1 \tag{4.10}$$

To compute the secant line $\ell_{T,Q}(P)$, recall that $\ell_{T,Q} = y - y_T - \frac{y_T - y_Q}{x_T - x_Q}(x - x_T)$. Using the notation above, and with a similar derivation to $\ell_{T,T}(P)$, we get a projective formula for $\ell_{T,Q}(P)$:

$$\ell_{T,Q}(P) = \lambda y_P - \theta x_P w + (\theta X_2 - \lambda Y_2)w^3.$$

We need to compute $\theta \cdot X_2, \lambda \cdot Y_2$ and $\lambda y_P, \theta x_P$. Note again that x_P is in $\mathbb{F}_p$, so λy_P costs $2m$, and similarly for θx_P. Therefore, the line evaluation costs an additional $2\tilde{m} + 4m$ when $T + Q$ is computed first.

Finally, we discuss a special case of multiplication in $\mathbb{F}_{p^{12}}$. We noted that line evaluations have the form $a+bw+cw^3$, where $a, b, c \in \mathbb{F}_{p^2}$, and in Algorithm 10, steps 4,6,8,12,13 all require multiplications of line functions. These elements are *sparse* compared to a generic element, and thus multiplications of the form $\ell \cdot f$ are simplified as follows. Let $f = f_0 + f_1 w$ where $f_0, f_1 \in \mathbb{F}_{p^6} = \mathbb{F}_{p^2}[n]/(v^3 - \xi)$, and write $\ell = a + bw + cw^3$ where $a, b, c \in \mathbb{F}_{p^2}$. Then, following [118], we have:

$$\begin{aligned} \ell \cdot f &= [a + (b + cv)w] \cdot (f_0 + f_1 w) \\ &= (af_0 + (b + cv)f_1 v) + (af_1 + (b_c v)f_0)w \\ &= (af_0 + (b + cv)f_1 v) + [(a + b + cv)(f_0 + f_1) - af_0 - (b + cv)f_1]w. \end{aligned}$$

There are two multiplications to be computed in the last expression above. Let $f_1 = f_{10} + f_{11}v + f_{12}v^2$. Then

$$\begin{aligned} (b + cv) \cdot f_1 &= (b + cv) \cdot (f_{10} + f_{11}v + f_{12}v^2) \\ &= (bf_{10} + cf_{12}\xi) + (cf_{10} + bf_{11})v \\ &\quad +[(b + c)(f_{10} + f_{11} + f_{12}) - bf_{10} - cf_{12} - cf_{10} - bf_{11}]v^2. \end{aligned}$$

Thus, the sparse multiplication is reduced to 5 multiplications in $\mathbb{F}_{p^2}$, namely $b \cdot f_{10}$, $c \cdot f_{12}$, cf_{10}, bf_{11}, and $(b + c) \cdot (f_{10} + f_{11} + f_{12})$. Note that multiplication by the constant ξ has negligible cost – an addition and a conjugation. The same factorization works for computing $(a + b + cv) \cdot (f_0 + f_1)$. Finally, $a \cdot f_0$ can be computed at a cost of $3\tilde{m}$ directly, so the total cost of a sparse multiplication is $5\tilde{m} + 5\tilde{m} + 3\tilde{m} = 13\tilde{m}$ (after discounting all additions, reductions, and negations).

We note that the first multiplication of the Miller variable with a line function is further simplified, as well as the multiplications in the adjustment steps 12,13 in Algorithm 10. We use the above estimates for simplicity, noting that the cost of the Miller loop is upper bounded by our final estimate and can be improved by accounting for additional optimizations.

Table 4.2 summarizes the costs of Miller loop operations that were estimated in this section. Note that the tangent and secant line evaluation table entries denote the *additional* cost required to compute the desired quantity, *after* the associated point addition or doubling inside the Miller loop, since the intermediate results are extensively reused. Recalling that $z = -(2^{62} + 2^{55} + 1)$, $6z + 2$ is a 65-bit integer of hamming weight 5, the cost of the Miller Loop is

$$ML = 64 \cdot (3\tilde{m} + 6\tilde{s} + 4m + 12\tilde{m} + 13\tilde{m}) + 4 \cdot (9\tilde{m} + 2\tilde{s} + 2\tilde{m} + 4m + 13\tilde{m});$$

$$ML = 1888\tilde{m} + 392\tilde{s} + 272m = 6720m.$$

This estimate upper bounds the count from [13], which reports $6504m$. This discrepancy comes from discounting one set of operations from the beginning of the Miller loop ($f = 1$), as well as special 'sparser' multiplication formulas.

Operation	Cost
$E'(\mathbb{F}_{p^2})$ doubling	$3\tilde{m} + 6\tilde{s}$
$\ell_{T,T}(P)$ evaluation	$4m$
$E'(\mathbb{F}_{p^2})$ point addition	$9\tilde{m} + 2\tilde{s}$
$\ell_{T,Q}(P)$ evaluation	$2\tilde{m} + 4m$
$\ell \cdot f$ sparse multiplication	$13\tilde{m}$
$\mathbb{F}_{p^{12}}$ squaring	$12\tilde{m}$

Table 4.2: Operation count for Miller loop operations

4.2.2 Final Exponentiation

The final exponentiation step consists of computing $f \mapsto f^{(p^{12}-1)/r}$ in $\mathbb{F}^*_{p^{12}}$. We can write this in the form

$$(p^{12} - 1)/r = [(p^{12} - 1)/\Phi_{12}(p)] \cdot [\Phi_{12}(p)/r],$$

where $\Phi_{12}(p) = p^4 - p^2 + 1$ is the 12th cyclotomic polynomial. The general idea is to split the exponent into two easy exponents and a hard exponent. The easy exponents take advantage of the p^{th} powering map. The hard exponent $(p^4 - p^2 + 1)/n$ is computed in the cyclotomic subgroup $G_{\phi_6}(\mathbb{F}_{p^2})$ (note that after exponentiating the intermediate pairing value f by $p^6 - 1$ and $p^2 + 1$, the result lies in $G_{\phi_6}(\mathbb{F}_{p^2})$). Recent work on efficient squaring methods in cyclotomic subgroups [69, 62, 107] focuses on handling this part of the exponentiation. Special squaring techniques discussed in [69] can be used to achieve the most efficient computation in this subgroup, and are used in highly optimized pairing implementations [13, 103].

Now, recall that p and n are both functions of an integer of low hamming weight, z (Section 4.1.4). Then it can be shown that the resulting polynomial has the following factorization in terms of p [48] :

$$\frac{p^4 - p^2 + 1}{n} = p^3 + (6z^2 + 1)p^2 + (36z^3 - 18z^2 + 12z + 1)p + (36z^3 - 30z^2 + 18z - 2).$$

The final exponentiation thus requires computing

$$(f^{p^3}) \cdot (f^{p^2})^{6z^2+1} \cdot (f^p)^{36z^3-18z^2+12z+1} \cdot f^{36z^3-30z^2+18z-2}.$$

To evaluate the above product, we can find a vector addition chain and do multiexponentiation, or exploit the form of the exponent and perform square-and-multiply. Devegili, Scott and Dahab [48] choose the latter option; see Algorithm 11. Note that p^e-th powering on an element of $\mathbb{F}_{p^{12}}$ reduces to powering in $\mathbb{F}_{p^2}$, as we show below. If we assume that all necessary powers of ξ are pre-computed and stored, only the products $a_i \cdot \xi^{(\cdot)}$ are required. Also note that since $\xi^i = 1$ for $i = 0$, the total number of $\mathbb{F}_{p^2}$ multiplications is 5. Then

$$\begin{aligned}
a^{p^e} &= \left(\sum_{i=0}^{5} a_i w^i\right)^{p^e} \\
&= \sum_{i=0}^{5} a_i^{p^e} w^{ip^e} \\
&= \sum_{i=0}^{5} a_i^{p^e} (w \cdot w^{p^e-1})^i \\
&= \sum_{i=0}^{5} a_i^{p^e} (w \cdot \xi^{(p^e-1)/6})^i \text{ since } p \equiv 1 \bmod 6 \\
&= \sum_{i=0}^{5} (a_i^{p^e} \cdot \xi^{i\frac{p^e-1}{6}}) w^i.
\end{aligned}$$

Thus, each p^e exponent mapping in Algorithm 11 essentially costs $5\tilde{m}$.

Algorithm 11 Hard exponentiation [48]

INPUT: $f \in G_{\Phi_{12}(p)} \subset \mathbb{F}_{p^{12}}^{\times}$,
OUTPUT: $f^{(p^4-p^2+1)/n}$

1: $a \leftarrow f^{6z-5}$
2: $b \leftarrow a^p$
3: $b \leftarrow a \cdot b$
4: Compute f^p, f^{p^2}, f^{p^3}
5: $f \leftarrow f^{p^3} \cdot [b \cdot (f^p)^2 \cdot f^{p^2}]^{6z^2+1} \cdot b \cdot (f^p \cdot f)^9 \cdot a \cdot f^4$
return f

We note that the work of Sanchez et al. makes use of Karabina's compressed squaring formulas, which are more efficient than the Stam and Lenstra formulas in Lemma 4.14.

For our first-order approximation, we use Algorithm 11 and the Stam-Lenstra formula for squaring in $G_{\Phi_{12}(p)}$. Denote the cost of squaring in the cyclotomic subgroup by S' and multiplication in $\mathbb{F}_{p^{12}}$ by M_{12}. Note that the integer $6z-5$ has length 65 and Hamming weight 7, and $6z^2+1$ has length 127 and Hamming weight 13. Because of the choice of z to have low Hamming weight, this makes square and multiply the most efficient exponentiation technique here, beating out window methods due to the necessary precomputations. Thus, we have $(64S' + 6M_{12}) + (126S' + 12M_{12})$ for special exponents in Algorithm 11. Exponentiations by 4 and 9 cost $2S' + (1M_{12} + 3S')$. There are 4 p^e-th powerings in $\mathbb{F}_{p^{12}}$, costing a total of $4 \cdot 5\tilde{m}$. Finally, Step 5 requires $1S' + 8M_{12}$ to combine the intermediate results. The easy exponentiations f^{p^2+1}, f^{p^6-1} cost $5\tilde{m} + M_{12}$ and I_{12} (inversion in $\mathbb{F}_{p^{12}}$), respectively. Inversion in $\mathbb{F}_{p^{12}}$ costs $97m + 1i$; this follows from the $\mathbb{F}_{p^6}$ inversion formula from [48]; see Chapter 5 of [118] for details. The cost of the final exponentiation is thus

$$FE = (196S' + 27M_{12} + 20\tilde{m}) + (5\tilde{m} + M_{12}) + I_{12}.$$

Putting this in terms of $\mathbb{F}_p$ operations yields

$$FE = (196 \cdot 2(2\tilde{m} + 3\tilde{s}) + 27 \cdot 18\tilde{m} + 20\tilde{m}) + (97m + 1i) = (196 \cdot 24 + 27 \cdot 54 + 60 + 97)m + 1i.$$

Hence $FE = 6319m + 1i$. The adjustment steps consist of 2 (line evaluation + point addition) steps, 2 sparse multiplications, 1 p-th power Frobenius endomorphism, and 1 p^2-power Frobenius endomorphism. The total cost for the adjustment steps is

$$2\tilde{m} + 4m + 9\tilde{m} + 2\tilde{s} + 2 \cdot (13\tilde{m}) + 6m + 2m = 127m.$$

If we assume a simple $a^{p-2} \pmod p$ inversion algorithm in $\mathbb{F}_p$ (with square and multiply), we find that computing a single optimal ate pairing costs approximately

$$\begin{aligned} ML \ + \ FE \ + \ \text{adjustment} &= 6720m + 6319m + 1i + 127m \\ &= 13166m + (127s + 95m) \approx 13388m, \end{aligned}$$

since p is a 256-bit prime of Hamming weight 95. This estimate differs from the reported count of $10152m$ from [13]; this is largely due to this analysis not taking all of the optimal choices that speed up the final exponentiation, e.g. even faster squaring in cyclotomic subgroups. Aranha et al. report that these formulas reduce the number of $\mathbb{F}_{p^2}$ squarings by 33%. This change alone accounts for approximately half of the discrepancy above.

4.3 Identity-Based Schemes

Identity-based cryptography was first proposed by Shamir in 1984 [109]. The idea is that the cost of key exchange is omitted, and the system simply uses an identifier (ID)

to encrypt messages and verify signed messages, as opposed to a public key certificate, which may have large components and can be costly to generate and maintain. The first identity-based key distribution, signature, and encryption schemes based on pairings are due to Sakai, Ohgishi and Kasahara [102]. The Boneh-Franklin identity-based encryption (IBE) scheme from 2001 was the first identity-based encryption scheme with a proof of security [29]. The Boneh-Franklin IBE led to an increase in research activity surrounding pairings for use in cryptography, and the trend has led to work on aggressive optimization of pairing computations for cryptographic applications [12, 103, 13].

The Boneh-Lynn-Shacham (BLS) signature scheme [31] is interesting in part due to its short signature: a signature on a message consists of a single group element. DSA and ECDSA signatures consist of two integers modulo n; concretely, this means 512 bit signatures vs. 256 bit signatures, at the 128-bit security level.

4.3.1 BLS Signature Scheme

In this section, we present the BLS signature scheme. BLS is interesting in the context of batch signature verification since we can use previously mentioned techniques for multi-scalar multiplication in G. There is also the possibility of doing multi-pairing computations, although the number of common operations is not sufficient to outperform batch verification in G. This agrees with the general understanding that computing a pairing is typically the most expensive part of any given protocol.

Gap Diffie-Hellman Groups

Let $G_1 = \langle g_1 \rangle, G_2 = \langle g_2 \rangle$ be finite groups of prime order p. Let $e : G_1 \times G_2 \to G_T$ be a Type 3 bilinear map. In the context of two groups, we can define generalized CDH and DDH as follows, following [31].

Computational co-Diffie-Hellman (co-CDH) on (G_1, G_2): Given $g_2, g_2^a \in G_2$ and $h \in G_1$ compute $h^a \in G_1$.

Decisional co-Diffie-Hellman (co-DDH) on (G_1, G_2): Given $g_2, g_2^a \in G_2$ and $h, h^b \in G_1$ output 'yes' if $a = b$ and 'no' otherwise. When the answer is 'yes' we say that (g_2, g_2^a, h, h^a) is a co-Diffie-Hellman tuple.

We note that if $G_1 = G_2$ (that is, the groups are not just isomorphic, they are represented with the same generator and element encoding), these definitions are just CDH and DDH in the single-group setting.

Now, in order to state the security theorem for BLS signatures, we will need a few definitions. Define the *advantage* [31] of an algorithm $\mathcal{A}$ in solving co-CDH on (G_1, G_2) as

$$\Pr[\mathcal{A}(g_2, g_2^a, h) = h^a : a \leftarrow_R \mathbb{Z}_p, h \leftarrow_R G_1)].$$

This advantage is interpreted as the probability that $\mathcal{A}$ can solve a random instance of the co-CDH problem, where the probability distribution is over the random bits accessed by $\mathcal{A}$, and over the elements of $\mathbb{Z}_p$ and G_1, each with equal probability. We say that an algorithm $\mathcal{A}$ (t, ϵ)-breaks the co-CDH on (G_1, G_2) if $\mathcal{A}$ runs in time at most t, and the advantage of $\mathcal{A}$ is at least ϵ.

Definition 4.15 (co-GDH pair [31])**.** Two groups (G_1, G_2) are a (t, ϵ)-Gap co-Diffie-Hellman pair if they satisfy the following properties:

1. The group operation on both G_1, G_2 can be computed in one time unit.
2. The co-DDH on (G_1, G_2) can be solved in one time unit.
3. No algorithm (t, ϵ)-breaks co-CDH on (G_1, G_2).

The first point in Definition 4.15 can be interpreted as an upper bound on running time, or as an 'efficiently computable' requirement for the group operations. The second and third points express the gap in problem difficulty, as well as a normalized time measurement under which no algorithm (t, ϵ)-breaks co-CDH on (G_1, G_2).

To see that pairing groups satisfy this definition, recall bilinearity: for all $u_1, u_2 \in G_1, v_1, v_2 \in G_2$, $e(u_1 + u_2, v_1) = e(u_1, v_1) \cdot e(u_2, v_1)$ and $e(u_1, v_1 + v_2) = e(u_1, v_1) \cdot e(u_1, v_2)$. Given this property, part 2 of Definition 4.15 is satisfied, since a co-DDH instance is solved by computing $e(h, g_2^a)$ and $e(h^b, g_2)$ and checking for equality. Note that, by bilinearity,

$$\frac{e(h^b, g_2)}{e(h, g_2^a)} = e(h, g_2)^{b-a}.$$

Hence $a \equiv b \pmod{p}$ if and only if the test for equality holds. The first and third requirements reduce to known properties of the bilinear CDH and normalizing operation cost. Thus, the groups on which bilinear pairings are defined give an example of Gap Diffie-Hellman group pairs. The authors of [31] note that the converse, i.e., the existence of gap Diffie-Hellman groups that come from a context other than bilinear pairings is an open problem. Given the above setting, we can describe the BLS signature scheme.

Key generation. Pick random $x \in_R \mathbb{Z}_p$. The secret key is x and the public key is $v = g_2^x$.

Signing. Given a secret key $x \in \mathbb{Z}_p$, and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(M) \in G_1$, and $\sigma \leftarrow h^x$. The signature on M is $\sigma \in G_1$.

Verification. Given a public key $v \in G_2$, a message $M \in \{0,1\}^*$, and a signature $\sigma \in G_1$, compute $h \leftarrow H(M) \in G_1$ and verify that (g_2, v, h, σ) is a valid co-Diffie-Hellman tuple. If so, output 'valid'; if not, output 'invalid'.

Boneh, Lynn and Shacham proved that this scheme is existentially unforgeable under adaptive chosen-message attack, assuming the hash function H is modelled by a random oracle [31] and that solving co-CDH in (G_1, G_2) is intractible.

4.3.2 BN-IBV Signature Scheme

We describe the identity-based verification (IBV) scheme due to Zhang et al [120]. It is derived from Π-Sig (CHP), a modification of the Camenisch-Lysanskya (CL) signature scheme [37]. We call this the BN-IBV signature scheme, since we will analyze it in the context of a particular BN curve. It was proved in [37] that under the LRSW assumption [38, 83] in G, the CHP signature scheme is existentially unforgeable in the random oracle model for message space $\mathcal{M} = \{0,1\}^*$. We give an updated security proof here in the Type 3 setting. The proof for the batch verification algorithm satisfying Definition 1.3 in CHP is given in [37].

CHP Security Proof

The CHP scheme in the Type 3 setting is as follows.

Let $e : G_1 \times G_2 \rightarrow G_T$ be a Type 3 bilinear pairing, where $G_1 = \langle g_1 \rangle$, $G_2 = \langle g_2 \rangle$ and G_T are cyclic groups of prime order q. Let T be a set of time periods that is "polynomially bounded in size", and let $|T| = \gamma$.

Let $H_1 : T \rightarrow G_1$ and $H_2 : T \rightarrow G_1$ be hash functions.
Let $H_3 : \mathcal{M} \times T \rightarrow \mathbb{Z}_q$ be a hash function.

Key Generation:

The private key is $x \in_R \mathbb{Z}_q$. The public key is $X_2 = g_2^x$.

Signing:

To sign a message m during time period $t \in T$, compute $a = H_1(t)$, $b = H_2(t)$, and $w = H_3(m, t)$. Then $\sigma = a^x b^{xw}$ is the signature on m with private key x.

Verification:

To verify a signature σ on message m for time period t, compute $a = H_1(t)$, $b = H_2(t)$, and $w = H_3(m, t)$, and check whether the following equation holds:

$$e(\sigma, g_2) = e(a, X_2) \cdot e(b, X_2)^w.$$

LRSW-3 Assumption. Suppose that $X_1 = g_1^x \in G_1$, $X_2 = g_2^x \in G_2$ and $Y = g_1^y \in G_1$ are given. Suppose also that there is an oracle $\mathcal{O}_{X_1,X_2,Y} : w \mapsto (a, a^y, a^{x+wxy}), a \in_R G_1$, which maps inputs $w \in \mathbb{Z}_q^*$ to random triples of this form. Then no p.p.t adversary $\mathcal{A}$ succeeds at producing a 4-tuple (w, t, t^y, t^{x+wxy}) (where m was not queried to the oracle) with any non-negligible probability.

Theorem 4.16 (CHP Security in Type 3)**.** *Under the LRSW-3 assumption in (G_1, G_2), the CHP-3 signature scheme is existentially unforgeable in the random oracle model for message space $\mathcal{M} = \{0,1\}^*$.*

Proof: Assume that there exists a p.p.t algorithm $\mathcal{F}$ (the forger) with the following properties:

1. $\mathcal{F}$ takes as input the public key X_2;
2. $\mathcal{F}$ is given access to random oracles H_1, H_2;
3. $\mathcal{F}$ is given access to random oracle H_3, to which it can make at most q_H queries for each time period t;
4. $\mathcal{F}$ is given access to a signing oracle Σ to which it can make at most one query for each time period t.

The goal for $\mathcal{F}$ is to output a valid signature for (m, t) where m was not queried to Σ in any time period $t \in T$.

Claim: There is a p.p.t algorithm $\mathcal{B}$ that interacts with $\mathcal{F}$ and answers $\mathcal{F}$'s hash function and signature queries, and outputs a solution to the LRSW-3 problem with non-negligible

probability.

Proof of claim:

Setup: $\mathcal{B}$ chooses a random $w^* \in \mathcal{M}$ and queries $\mathcal{O}_{X_1,X_2,Y}(w^*)$ to obtain an LRSW-3 instance $(a^*, b^*, c^*) = (a^*, (a^*)^y, (a^*)^{x+w^*xy})$, for a randomly chosen a^* by the oracle. Select a random index $j^* \in T$, which will denote the time period for which we hope that $\mathcal{F}$ produces a valid forgery.

Key Generation: $\mathcal{B}$ supplies the public key $X_2 = g_2^x$ to $\mathcal{F}$.

Oracle Queries: For each time period $j, j \neq j^*$, $\mathcal{B}$ selects random pairs of integers $(r_j, s_j) \in \mathbb{Z}_q \times \mathbb{Z}_q$.

For the query that $\mathcal{F}$ makes to H_1 in time period j, $\mathcal{B}$ responds with

$$H_1(j) = \begin{cases} g_1^{r_j} & \text{if } j \neq j^* \\ a^* & \text{if } j = j^* \end{cases}$$

and for H_2, $\mathcal{B}$ responds with

$$H_2(j) = \begin{cases} g_1^{s_j} & \text{if } j \neq j^* \\ b^* & \text{if } j = j^* \end{cases} .$$

$\mathcal{B}$ also selects a random index $k^* \in [1, q_H]$, which represents the only hash query for which $\mathcal{F}$ queries the signing oracle in time period j^*. $\mathcal{B}$ selects $t_{j,k} \in_R \mathbb{Z}_q$, for each $k \in [1, q_H]$ and $j \in [1, \gamma]$. For queries to H_3, $\mathcal{B}$ responds with

$$H_3(m_k, j) = \begin{cases} t_{j,k} & \text{if } j \neq j^* \text{ or } k \neq k^* \\ w^* & \text{otherwise.} \end{cases}$$

For signing queries, $\mathcal{B}$ responds with the following:

$$\text{Sign}(m_k, j) = \begin{cases} \text{abort} & \text{if } j = j^* \text{ and } k \neq k^* \\ c^* & \text{if } j = j^* \text{ and } k = k^* \\ X_2^{r_j} X_2^{(s_j)t_{j,k}} & \text{otherwise.} \end{cases}$$

Note the reason for these responses: in the first case, $\mathcal{B}$ has no way of responding to such a signing query, as it would require knowledge of the private key. In the second case, $\mathcal{B}$ can supply a valid response because in this case the valid signature is given by the LRSW-3

instance (a^*, b^*, c^*), and since $\mathcal{B}$ responds to H_3 queries in such a way that $\mathcal{F}$ will get w^* when queried for time period j^* and message m_{k^*}.

Let σ be the forged signature on (m_k, j) produced by $\mathcal{F}$. If $j \neq j^*$ or $k = k^*$, then $\mathcal{B}$ outputs a random guess for the LRSW-3 challenge. If $\mathcal{B}$ is not forced to abort or issue a random guess, then we note that

$$\sigma = H_1(j^*)^x H_2(j^*)^{x \cdot H_3(m_k, j^*)}.$$

We have that $H_3(m_k, j^*) = t_{j^*,k} \neq w^*$. We can substitute $\sigma = (a^*)^x (b^*)^{x \cdot t_{j,k}}$, so $(t_{j,k}, a^*, b^*, \sigma)$ is a valid LRSW-3 instance. Thus $\mathcal{B}$ succeeds whenever $\mathcal{F}$ successfully forges a signature (and $\mathcal{B}$ is not forced to abort). ■

The analysis of the success probability for $\mathcal{B}$ is the same as in the Type 1 setting, so we omit it here. Note that since q_H and γ may be large, this security proof is considerably non-tight. We also note that the proof in Type 3 is virtually identical to the proof in the Type 1 setting, except for the pairing groups and their generators. However, there is a choice made when modifying the scheme: we chose the hash functions to map into G_1 and for public keys to be in G_2. It is also possible to define the scheme with public keys in G_1, but since elliptic curve computations are fastest in G_1 in practice, and hashing is also faster in that setting, this seems to be the obvious choice for efficiency and shortest signatures. This has the side effect of longer public keys, but in the context of vehicular ad-hoc networks this is a favourable trade-off.

Zhang et al. designed their signature scheme with several desirable privacy and security properties. It has been noted in the literature that user privacy is required, but it should be possible to hold misbehaving units accountable. User anonymity is achieved through pseudo-identities, which are simply random elements of G_1. If desired, pseudo-identities can be changed as frequently as is practical, since these multiples can be precomputed offline. This defeats traffic analysis since messages sent by a vehicle cannot be linked to messages sent under a different pseudo-identity. However, this looks different from the point of view of the trusted authority since it knows the master private key and can recover the long-term public key of each unit. The trusted authority (TA) may determine the *real identity*, RID of the malicious unit with known quantities, along with its master secret (s_1, s_2):

$$ID_2 \oplus H(s_1 ID_1) = RID \oplus H(rP_{Pub1}) \oplus H(s_1 rP_1) = RID.$$

Let $e : G_1 \times G_2 \to G_T$ be a bilinear, nondegenerate Type 3 pairing. Zhang et al. define the scheme using a Type 1 pairing. We are interested in the Type 3 setting, so we will have to modify the scheme to make the point $s_1 P_2 \in G_2$ part of the public key. This modification simply means publishing an additional public key in G_2.

Let $G_1 = \langle P_1 \rangle, G_2 = \langle P_2 \rangle$, and let $H : \{0,1\}^* \to G_1$, $h : \{0,1\}^* \to \{0,1\}^\ell$, where we leave ℓ dependent upon context and the desired security level. We also assume that all signatures that we batch verify are from the same time/location period Φ, typically realized as a time interval between 100 and 300 milliseconds in practice [120]. Each road side unit (RSU) must be capable of verifying all message-signature pairs in this time interval.

Key Generation and Pre-Distribution
The TA chooses $s_1, s_2 \in_R \mathbb{Z}_q^*$, called *master keys*, for a vehicle.
The TA computes $P_{pub1} = s_1 P_1$, $P_{pub2} = s_2 P_2$, $P_{pub3} = s_1 P_2$.
Each RSU and vehicle gets $\{G_1, G_2, G_T, q, P_1, P_2, P_{pub1}, P_{pub2}, P_{pub3}\}$.
Each vehicle gets $\{s_1, s_2\}$ in a tamper proof unit (see Figure 4.1). To access the tamper proof unit, each vehicle has its own real identity, RID, and password PWD. Note the difference between the original scheme and the one presented here: P_{pub3} is required here.

Pseudo-Identity and Private Key Generation
Authentication module: Requires the correct RID and PWD to access.

Pseudo-Identity Generation Module:
$ID_1 = rP_1 \in G_1$, where $r \in_R [1, q-1]$
$ID_2 = RID \oplus H(rP_{pub1})$

Private Key Generation Module:
$SK_1 = s_1 ID_1 \in G_1$
$SK_2 = s_2 H(ID_1 || ID_2) \in G_1$

Signing
i) Vehicle V_i generates traffic related message M_i.
ii) V_i picks pseudo-identity $ID^i = (ID_1^i, ID_2^i)$ and the corresponding private key $SK^i = (SK_1^i, SK_2^i)$ by way of the tamper-proof device.
iii) With the private key, V_i computes signature σ_i of message M_i, where

$$\sigma_i = SK_1^i + h(M_i) SK_2^i \in G_1.$$

iv) V_i sends $\langle ID^i, M_i, \sigma_i \rangle$ to an RSU.
v) These steps are performed once per public key per time/location interval Φ.

Single Signature Verification
Given $\{G_1, G_2, G_T, q, P_1, P_2, P_{pub1}, P_{pub2}, P_{pub3}\}$, message $\langle ID^i, M_i, \sigma_i \rangle$, the signature σ_i is valid if

$$e(\sigma_i, P_2) = e(ID_1^i, P_{pub3}) \cdot e(h(M_i) H(ID_1^i || ID_2^i), P_{pub2}).$$

Remark 1. Note that BN-IBV is loosely an instantiation of CHP, since $H_3(m||j)$ corresponds to $h(M)$, H_2 corresponds to $H(ID_1||ID_2)$, and H_1 corresponds to $r \cdot P_1$. However, note that in CHP there is one secret key, x_j, but in BN-IBV there are s_1 and s_2. It is an open problem to define a security model and a reductionist security proof for BN-IBV.

Batch Verification

Suppose we have N distinct messages denoted $\langle ID^1, M_1, \sigma_1\rangle, \ldots, \langle ID^N, M_N, \sigma_N\rangle$. If all N signatures are valid, then

$$e\left(\sum_{i=1}^{N}\sigma_i, P_2\right) = e\left(\sum_{i=1}^{N} ID_1^i, P_{pub3}\right) \cdot e\left(\sum_{i=1}^{N} h(M_i)H(ID_1^i||ID_2^i), P_{pub2}\right). \tag{4.11}$$

Remark 2. Equation 4.11 alone does not suffice for a batch verification algorithm. In order to prove that all N equations hold with error probability $2^{-\ell}$, we must use the small exponents test. Equation 4.11 is presented by Zhang et al. as a batch verifier, but this is not justified in [120]. In particular, it is not at all clear whether Equation 4.11 implies that each of the N signature verification equations is satisfied. Therefore, in order to remedy this, we will select random scalars $\delta_i \in \{0,1\}^{\ell}$ for $i = 1, \ldots, N$, perform the small exponents test, apply the pairing, and report operation counts for *this* batch verifier. In this case, the batch verifier proof carries over from Π-sig in [37]. Thus, we much check that

$$e\left(\sum_{i=1}^{N}\delta_i\sigma_i, P_2\right) = e\left(\sum_{i=1}^{N} \delta_i ID_1^i, P_{pub3}\right) \cdot e\left(\sum_{i=1}^{N}(\delta_i \cdot h(M_i) \mod q)H(ID_1^i||ID_2^i), P_{pub2}\right).$$

Since batch verification requires just 3 pairing computations, amortizing the cost of these pairings over the batch size is once again the overall goal. Rodriguez-Henriquez et al. note that the cost of group operations in G_1 and G_2 is more relevant now that pairing computations have become more efficient [103].

Despite its computational benefits, batch verification is a natural target for denial of service attacks. Once a certain threshold (depending on signature scheme, primitives used, etc. up to a given theoretical bound [119]) of invalid signatures is present in a batch, we must fall back on individual verification. As a result, a lot of computation is wasted because multi-exponentiations performed during batching signatures do not all help with verifying individual signatures, and this loss of efficiency is enough to cause a denial of service.

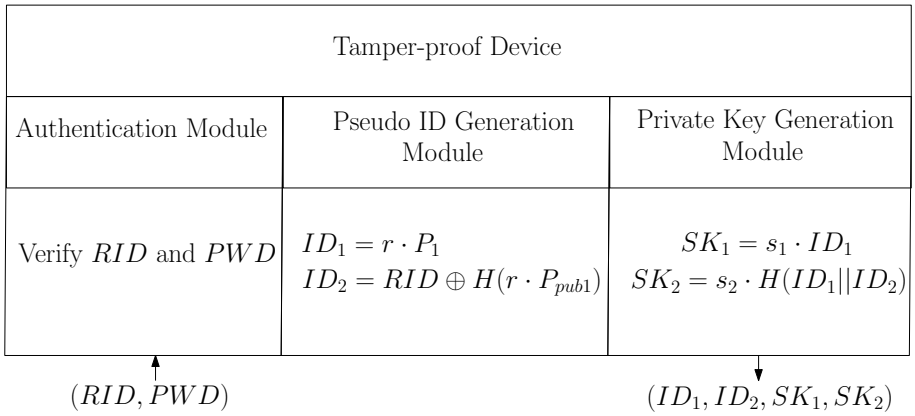

Figure 4.1: The tamper-proof device [120]. The only modification required in the Type 3 setting is letting $ID_1 = r \cdot P_1$.

4.4 Comparing Conventional ECC with Pairings

In this section, we compare the cost of BN-IBV batch verification instantiated with the BN curve from Section 4.3, batch verification of EdDSA signatures instantiated with Ed25519 and Bos-Coster batching (Chapter 3), and Cheon-Yi batching with the ECDSA* signature scheme (Chapter 2). All comparisons are at the 128-bit security level. The following analysis considers schemes already presented in this thesis and evaluates their respective operation cost and features.

4.4.1 BN-IBV

The cost of a single signature verification in the Zhang et al. scheme can be computed as follows. The RSU receives a tuple of the form (ID, M, σ), and must verify that

$$e(\sigma, P_2) = e(ID_1, P_{pub3}) \cdot e(h(M)H(ID_1||ID_2), P_{pub2}).$$

This requires: one hash function evaluation on M (which is discounted because we assume short messages), 1 map-to-point hash on $ID_1||ID_2$, 1 scalar multiplication in G_1, 3 optimal ate pairing evaluations, and 1 multiplication in $\mathbb{F}_{p^{12}}$.

BN-IBV uses hashing into G_1, so the means of accomplishing this task requires some attention. However, hashing into pairing groups is an interesting problem in itself [54]. We will take the hash function for granted and discount the computational cost, just as we do

in the case of EdDSA. Since the map-to-point hash is evaluated for every pseudo-identity (which may change with every message), it requires variable-base scalar multiplication. We use the w-NAF method (discussed in Section 2.3.2) to compute $h(M)H(ID_1||ID_2) \in G_1$, as suggested in Section 4.1 of Sanchez et al. [103] (since $h(m)$ is a random scalar, verification benefits from windowing).

We assume that the RSU operates on decompressed points: in practice, transmitting points in compressed format is common and desirable, but here we want to focus on the verification costs. We will assume that each RSU has Jacobian coordinate representations of intermediate group elements in computation and affine representation for any pre-computed points, e.g. pseudo-identities. Compression and decompression of points and keys in EdDSA is part of the actual signature scheme, so we will assume that both BN-IBV and ECDSA* use standard compression of points, noting that they are not widely different in cost according to Section 5 of [25]. For our purposes, verification costs implicitly include the cost of compression and decompression, which are approximately the same in both the Weierstrass and Edwards cases.

We use the w-NAF with online precomputation for $w \in \{3, 5\}$, where m denotes the bit-length of the group order. This gives a variable-base scalar multiplication cost of

$$[1D + (2^{w-2} - 1)A] + \left[\frac{m}{w+1}A + mD\right].$$

This can be further improved by using a 2-dimensional decomposition of the scalar (due to Gallant-Lambert-Vanstone), which changes the online doubling cost to $\frac{m}{2}D$ from mD. Note that we cannot take advantage of mixed addition formulas here because the precomputation is *online* and thus converting those points to affine would only further increase the cost (over and above the slow Jacobian full additions). Hence, variable-base scalar multiplication costs

$$[(3m + 4s) + 7 \cdot (12m + 4s)] + \left[\frac{256}{5+1}(12m + 4s) + \frac{256}{2}(3m + 4s)\right] = 983m + 711s.$$

There are additional techniques for reducing pairing costs in the context of products of pairings (i.e. multi-pairing). Namely, the accumulator f in Algorithm 10 can be shared among the Miller loops for each individual pairing. In addition, we can take advantage of precomputation whenever there is a fixed point $Q \in G_2$ for which we evaluate the pairing. A multi-pairing algorithm taking these improvements into account is given in [103] (Algorithm 8).

Suppose we are to evaluate a single optimal ate pairing. If the argument $Q \in G_2$ is fixed then we precompute line functions $\ell_{T,T}(\cdot)$, as well as the point multiples $\{Q, [2]Q, [2^2]Q, \ldots, [2^{64}]Q\}$, at the cost of extra storage (65 G_2 elements, $64+4 = 68$ line functions). At run-time, they are evaluated at the variable point $P \in G_1$. More concretely: the line function $\ell_{T,T}(\cdot) = \ell_0 y_P + \ell_1 x_P w + \ell_2 w^3$ will have the ℓ_i precomputed offline. The additional cost remains multiplying the results by x_P and y_P; we actually only save the cost of all the point doublings in the Miller loop. The same is true for $\ell_{T,Q}(\cdot)$, and the additional cost of line function evaluation is reduced to $4m$ in this case. Hence

$$ML = 64 \cdot (0 + 4m + \underline{12\tilde{m}} + 13\tilde{m}) + 4 \cdot (0 + 4m + 13\tilde{m}) = \underline{768\tilde{m}} + 884\tilde{m} + 272m,$$

$$FE = 6319m + 1i.$$

The underlined portions denote *shared* operations in the multipairing context, so they are performed only once in a product of pairings. The remaining (non-underlined) terms are repeated for each individual pairing in the product. The adjustment steps are also repeated for each individual pairing in the product, and the final exponentiation cost is shared among *all* the pairings, so the FE count from Section 4.2.2 is unchanged. The total pairing cost for single verification is

$$2 \times \underline{768\tilde{m}} + 3 \times 884\tilde{m} + 3 \times 272m + 2 \times \underline{FE} + 3 \times \text{adjustment} = 26589m + 254s \approx 26843m.$$

Therefore, the cost of a single verification for BN-IBV is approximately $26843m + 983m + 711s + 18\tilde{m} \approx 28591m$. Note that this is significantly less than the cost of three optimal ate pairings, relying heavily on pre-computation and interleaving operations.

As expected, the contribution of each pairing required for verification makes the overall cost quite high. However, the cost per signature reduces significantly in the case of batching. Recall the batch verification equation for BN-IBV:

$$e\left(\sum_{i=1}^{N} \delta_i \sigma_i, P_2\right) = e\left(\sum_{i=1}^{N} \delta_i ID_1^i, s_1 P_2\right) \cdot e\left(\sum_{i=1}^{N} (\delta_i \cdot h(M_i) \mod q) H(ID_1^i || ID_2^i), P_{pub2}\right).$$

Verifying this equation requires N modular multiplications (modulo the curve order), 1 $\mathbb{F}_{p^{12}}$ multiplication, 3 multi-scalar multiplications, and 3 optimal ate pairings (note the product of pairings and fixed arguments in G_2). Assuming we use the Bos-Coster algorithm for $N = 64$, the cost for multi-scalar multiplication is approximately 1650 (full) Jacobian-coordinate additions. We choose to present operation counts with a concrete power of two, and later note the asymptotic cost as a function of N in Table 4.4. The output of the map-to-point hash from [54] is in affine coordinates (i.e. the $H(ID_1^i || ID_2^i)$ are all affine),

but Algorithm 8 requires mostly projective additions. The first few steps in the Bos-Coster algorithm may be mixed additions, but the sorted points A_i all eventually become affected by the algorithm and thus in projective coordinates with $Z \neq 1$. Analysis on how quickly this happens on average could be interesting future work, since the savings in using mixed addition formulas is desirable, and will enable an estimate that is a tighter upperbound than what we accomplish here. Using Jacobian additions ($12m+4s$ according to Table 2.2), the total cost of a batch verification in BN-IBV as

$$\text{Batch}_N(\text{BN-IBV}) \approx 3 \times (\text{B-C})_N \times (\mathcal{A}_e) + 26843m + 18\tilde{m} + Nm_q.$$

Thus $\text{Batch}_{64}(\text{BN-IBV}) \approx 106161m$, using Table 3.5 for the cost of Bos-Coster for $N = 64$ and assuming that modular multiplications modulo the curve order cost the same as field multiplications ($m_q = m$), and full projective additions ($\mathcal{A}_e = 12m + 4s$). We note that the BN-IBV scheme is sensitive to invalid signatures. We have discussed the issue of batch forgery identification, and the problem has been explored by Matt and Law in [81, 84]. If one expects to encounter a large number of invalid signatures in a batch, the cost of verification increases significantly due to falling back on single signature verification.

Operation	Cost
Variable base scalar multiplication in G_1	$983m + 711s$
Single O-Ate Pairing on $G_1 \times G_2$	$13388m$
BN-IBV Batch Verification ($N = 64$)	$\approx 106161m$

Table 4.3: Pairing operation costs summary

4.4.2 EdDSA

Single signature verification in the EdDSA scheme is computed as follows. After decompression, compute one join sparse form and check whether $R = SB - H(\underline{R}, \underline{A}, M)A$. In fact, one checks whether the *encoding* of R is the same as the encoding of the right hand side of this verification equation. This saves the cost of decompressing R, and is the main reason that the technique of Antipa et al. is not of clear benefit in this scenario since it requires decompressed R. ECDSA* already operates on decompressed points, so it can take advantage of half-length scalars. However, one is forced to use larger keys and signatures in ECDSA* if using decompressed points. The double-scalar multiplication method used is not specified explicitly in [25]. Sliding windows are one natural choice [106], and JSF is another reasonable choice [64]. We choose extended twisted Edwards coordinates, which require $9m + 2d$ for a point addition ($8m + 1d$ for mixed, -1 twist), in the notation of

Table 3.4, and the details are given in Algorithm 3.4.1. Doubling in extended twisted Edwards coordinates requires $3m + 4s$ according to Section 3.3 of [67] (formulas independent of d, and -1 twist simplifies formulas). After discounting (offline) pre-computation on the base point B, an EdDSA single verification using the JSF costs

$$\left(\frac{253}{2} + 2\right) A + 253D = 126.5 \times (8m + 1d) + 253 \times (3m + 4s) = 1771m + 1012s + 127d.$$

To aid in comparison with other schemes, s/m ratios are estimated between 0.8 and 1 in practice, and so far we have taken $s/m = 1$. Determining the ratio d/m for a twisted Edwards curve with $d = -121665/121666$ must be measured in practice. Bernstein et al. note that one could trade this multiplication for a small number of multiplications by 121665 and 121666, but the software does not do this to save code size. Taking $d/m = 1$ is a conservative choice, but in the absence of other data it will suffice. Thus a single verification costs approximately $2910m$ in the worst case. We should add $90m$ for a single inversion of the Z coordinate allowing us to use the mixed addition formulas, giving approximately $3000m$ for single verification.

Batch Verification

We have analyzed the Bos-Coster Algorithm (Algorithm 8) in Section 3.5.2. In Table 3.5, we gave a concrete operation count in terms of elliptic curve additions and modular subtractions. Recalling the batch verification equation 3.2 for EdDSA:

$$\left(-\sum_i z_i S_i \mod l\right) B + \sum_i z_i R_i + \sum_i (z_i H_i \mod l)\, A_i = 0,$$

there is one fixed-base scalar multiplication for the base point B with a full-length scalar, one multi-scalar multiplication involving uncompressed public keys A_i (compare to: Q_i for ECDSA*) with 253-bit scalars, and one multi-scalar multiplication on uncompressed signature points R_i with 128-bit scalars (compare to Section 5 of [25]).

Curve25519 accomplishes fixed-base scalar multiplication in signing and verification with lookup tables. During signing, one must ensure constant-time computation and avoid branch conditions to protect against side-channel attacks, so care is needed when computing rB [25]. During verification, there is no need to protect against side-channel attacks because the scalar is known to anyone listening to message-signature broadcasts, and the fastest, non-constant time algorithm may be used to verify. Thus, we use a sliding window w-NAF. Since the cost of Bos-Coster is directly proportional to the input length of the

scalars (for a fixed batch size n), a 253-bit multi-scalar multiplication costs approximately $1650 \times \frac{253}{128} = 3262A$. Note that we can only use mixed addition formulas for the fixed-base computation. The cost of batching 64 EdDSA signatures with the Bos-Coster algorithm is thus approximately

$$\begin{aligned}
\text{Batch}_{64}(\text{EdDSA}) &= 1650 \cdot A + (1 \text{ fixed window}) \ + \ (3262 \cdot A) \\
&= 1650A + \lceil 253/5 \rceil (8m + 1d) + 3262A \\
&= 4912(9m + 1d) + 51(8m + 1d) \\
&= 44616m + 4963d \\
&\approx 49579m,
\end{aligned}$$

in the worst case.

Recalling the remarks in Section 2.2 and 2.3.2, we consider again the question of whether the Bos-Coster method and the acceleration technique of Antipa et al. are orthogonal improvements. Since in this case our batch verification algorithm does not rely on the chosen sparseness of scalars, we could make the expensive lattice computation every time we perform a batch verification in order to get (almost all) half-length scalars. Since Bos-Coster requires only additions, a first order analysis of the trade off is straightforward. In order to make the method of Antipa et al. worthwhile in this context, the lattice computation must be less than the incremental cost. In other words,

$$(\text{short vector cost}) + (\text{batching with half length scalars}) \leq 49579m.$$

After computing the cost of batching with half length scalars, we see that

$$(\text{short vector cost}) \leq 49579m - (51 \times 9m + 4912/2A) = 24560m.$$

Hence, finding short vectors in a 64-dimensional lattice must be cheaper than 24560 field multiplications. Future work may include a closer look at whether current implementations of lattice algorithms can attain this performance.

4.4.3 ECDSA*

Recall our discussion of ECDSA* from Chapter 2. Equation 2.20. To attain approximate 128-bit security, we chose parameters $m = 252$, $w = 5$, $t = 14$ in Section 2.3.2. Recalling equation 2.20, we have online precomputation and thus general Jacobian additions and

doublings are required. We make no assumptions about the ability of an RSU to store precomputed multiples of public keys, and note that if this changes in the future, the estimates in this section can be updated with the cost of mixed Jacobian-affine addition formulas. Finally, recall equation 2.23:

$$\frac{128}{3}\mathcal{A}_e + 256\mathcal{D}_e + 72.7N\mathcal{A}_e + 2N\mathcal{D}_e.$$

This gives a batch verification cost with $N = 64$ of

$$\begin{aligned}\text{Batch}_{64}(\text{ECDSA}^*) &= \left(\frac{128}{3} + 72.7N\right)A + (256 + 2N)D \\ &= (4695.5) \times (12m + 4s) + (256 + 128) \times (4m + 4s) \\ &= 76416m,\end{aligned}$$

assuming $s/m = 1$ in the worst case.

Remark 3. Noting that twisted Edwards curves with $a = -1$ give very efficient group law computation, the same calculation in the twisted Edwards case would give approximately $45000m$. This seems to suggest that these batching algorithms are similar for batch sizes of interest (although not asymptotically). We can consider other combinations, for instance ECDSA* with Bos-Coster and Weierstrass curves, $N = 64$; counting group operations as in Section 4.4.2, this costs $4912 \times (12m + 4s) + 51 \times (8m + 3s) = 79153m$. ECDSA* with Bos-Coster and twisted Edwards curves has the same cost of batch verification for $N = 64$ as EdDSA, namely $49579m$.

4.4.4 Comparison

We choose several points of comparison between signature schemes and present them in this section. Table 4.4 gives an overview; Bos-Coster and Cheon-Yi are abbreviated as B-C and C-Y, respectively. The choice of whether the scheme uses compression or not is denoted by χ, which affects the signature, public key, and certificate size (if applicable). The difficulty of Pollard Rho estimates come from Safecurves [4]. However, not all comparisons can be made directly. For instance, the fields used by each context are different (e.g. $GF(2^{255}-19)$ vs. $GF(p(z))$) and while $GF(2^{255}-19)$ can be made to use extremely efficient reduction and arithmetic algorithms, $GF(p(z))$ must fall back on generic Montgomery methods, which perform well enough, but cannot hope to match the same speeds. Interpreting operation counts should take this into account, but a fine analysis of field arithmetic is out of scope for this thesis.

The first order analysis offered here means to suggest that a trade-off is possible in terms of cost per-signature for *identity privacy preservation*. Sometimes the word *conditional* is a prefix to this phrase, denoting that it should be possible for the trusted authority to identify misbehaving participants, but not by, e.g. law enforcement (without a court order) [98]. Raya and Hubaux analyze this property in Section 7.2 of [98], and note that to maintain anonymity, the OBU should change their public key (or pseudo-identity in the identity-based case) roughly every minute. In a PKI this is accomplished with Certificate Revocation Lists (CRLs) and certificate expiry. Each message-signature pair that verifies means that it comes from a user that was trusted at *some* point, and has not misbehaved using their credentials before the CRL is updated. At some later point in time, their certificate may be revoked or expire, and their messages will cease to be trusted by users checking the CRL. The solution in the PKI is for each vehicle to store a large number of certificates and keys. This requires 525600 certificates per year, per vehicle to maintain identity privacy. This is a much larger requirement over time than any other precomputation involved in the BN-IBV scheme, but is not unreasonable storage by modern standards: with 257-bit ECC certificates [7], this requires approximately 128 MB (2^{20} bytes) of storage per year of use in a single vehicle. Practical considerations may not allow for a large amount of storage, and the flexibility of changing identities on the fly with BN-IBV, at a variable rate if desired, may be an independently attractive feature. One current proposal lists an area of future work in a PKI-based solution is to determine how many simultaneously valid (static) public key certificates a vehicle must use to maintain privacy [117].

	BN-IBV + B-C	ECDSA* + C-Y	EdDSA + B-C
Signature Size	$32 \times (1 + \chi)$ bytes	$\lvert(R, s)\rvert = 32 \times (2 + \chi)$ bytes	$\lvert(\underline{R}, \underline{S})\rvert = 64$ bytes
Certificate Size	0 (N/A)	$32 \times (2 + \chi)$ bytes	64 bytes
Key Size	$32 \times (1 + \chi)$ bytes	$\lvert Q\rvert = 32 \times (1 + \chi)$ bytes	$\lvert\underline{A}\rvert = 32$ bytes
Pollard Rho Difficulty	$2^{126.4}$	$2^{125.99}$	$2^{125.8}$
Batching Cost ($N = 64$)	$106161m$	$76416m$	$49579m$
De/Compression	$\chi \in \{0, 1\}$	$\chi \in \{0, 1\}$	Edwards
Identity Privacy	Flexible	Fixed cert. store	Fixed cert. store

Table 4.4: Comparison of signature schemes for vehicular ad-hoc networks

We summarize our discussion in this section with Table 4.4. Key size in the BN-IBV entry of Table 4.4 means the size of a pseudo-identity, which is a single (compressed) G_1 element.

BN-IBV is expensive in the single signature case, while a single JSF with EdDSA requires at least 10 times fewer operations. Since the purpose of this section is to investigate

how much this factor is diminished when considering the case of batching, where we consider the cost of verification *per signature batched*. This means one might be willing to take a *slight* performance hit per signature, while gaining Identity Privacy Preservation and smaller communication overhead. Dividing the table entries under "batching cost" by $N = 64$ gives:

$$\frac{\text{Batch}_{64}(\text{BN-IBV} + \text{B-C})}{64} = 1659m/\text{sig}$$
$$\frac{\text{Batch}_{64}(\text{ECDSA}^* + \text{C-Y})}{64} = 1194m/\text{sig}$$
$$\frac{\text{Batch}_{64}(\text{EdDSA} + \text{B-C})}{64} = 775m/\text{sig}.$$

The asymptotic batching costs are given in Table 4.5. Because of greater variation in curve parameters and associated costs for Edwards curves, we do not introduce additional notation and simply give the field operation costs for the third table entry. The constant C is determined in practice by experiment, e.g. Table 3.5.

BN-IBV+B-C	$3 \times (\text{B-C})_N \times \mathcal{A}_e + N \cdot m_q + 18\tilde{m} + a_{opt}$
ECDSA* + C-Y	$\left(\frac{128}{3} + 72.2N\right) A_{\text{full}} + (256 + 2N)D$
EdDSA + B-C	$(\text{B-C})_N \times (1 + \frac{253}{128}) \times (9m + 1d) + \lceil \frac{253}{5} \rceil (8m + 1d)$

Table 4.5: Asymptotic batching cost, 128-bit security

We note once again that field multiplications are not necessarily comparable in the above estimates - the effect of fast reduction algorithms is important in practice [13, 25] and we have not carried through the cost of modular reduction in this analysis. Future work would include comparing the implementations used in practice, while using the presented batching algorithms in order to verify the accuracy and applicability of our cost estimates.

Zhang et al. [120] evaluated the BN-IBV scheme in the Type 1 setting, using MNT curves. Although no concrete performance numbers are given in that paper, the basic asymptotic cost is examined, and the authors note that "IBV is 35.6% faster than ECDSA" at verifying signatures. The comparison presented there only considers a single verification algorithm for plain ECDSA (no batching). In our analysis, we have shown that just using Cheon-Yi batching will outperform single verification significantly, i.e. by a factor of 2.6. We have also considered more recent work in signature schemes and developments in elliptic curve cryptography, and will hopefully encourage further investigation. ECDSA* with twisted Edwards curves would essentially close the batching verification cost gaps. The main points of comparison are considered for various batch sizes N in Table 4.6.

Note that the doubling and addition cost used for Cheon-Yi batching and ECDSA* are different from Chapter 2: here we make no assumptions about pre-computation, so the projective additions are counted as full and not mixed, unless explicit account is made for Z coordinate inversions to allow faster formulas, and a single JSF computation is assumed throughout. EdDSA has single verification cost $3000m$, and ECDSA* has single verification cost 3568 for Weierstrass, and 3052 for twisted Edwards if the group order is 256 bits; if we use the twisted Edwards curve isomorphic to curve25519 then single verification costs $3000m$ for both. Results in the twisted Edwards cases are given assuming the same curve is used for both. Being more or less conservative affects the ratios of batch verification to single verification, but not the cost per signature. It is not difficult to re-produce Table 4.6 for varying estimates of single verification depending on the level of allowed precomputation.

N	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}
BN-IBV + B-C	9.94	5.58	3.43	2.29	1.66	1.28	1.05	0.89	0.78	0.70	0.63
ratio	0.348	0.195	0.120	0.080	0.058	0.045	0.037	0.031	0.027	0.024	0.022
ECDSA* + C-Y	1.86	1.52	1.35	1.26	1.22	1.20	1.19	1.18	1.18	1.18	1.18
ratio	0.522	0.426	0.378	0.354	0.336	0.333	0.332	0.331	0.331	0.331	0.331
ECDSA* + B-C	3.35	2.28	1.78	1.47	1.27	1.07	0.95	0.84	0.76	0.68	0.62
ratio	0.938	0.639	0.499	0.411	0.349	0.301	0.265	0.236	0.212	0.192	0.175
ECDSA* + C-Y (ted)	1.30	1.02	0.88	0.81	0.78	0.76	0.75	0.75	0.74	0.74	0.74
ratio	0.432	0.339	0.293	0.270	0.259	0.253	0.250	0.248	0.248	0.247	0.247
ECDSA* + B-C (ted)	2.10	1.43	1.11	0.91	0.77	0.67	0.59	0.52	0.47	0.42	0.39
ratio	0.701	0.476	0.370	0.304	0.258	0.222	0.195	0.174	0.156	0.141	0.129
EdDSA + B-C	2.10	1.43	1.11	0.91	0.77	0.67	0.59	0.52	0.47	0.42	0.39
ratio	0.701	0.476	0.370	0.304	0.258	0.222	0.195	0.174	0.156	0.141	0.129
EdDSA + C-Y	1.29	1.02	0.88	0.81	0.78	0.76	0.75	0.75	0.74	0.74	0.74
ratio	0.430	0.339	0.293	0.270	0.258	0.253	0.250	0.248	0.248	0.247	0.247

Table 4.6: Batch verification cost (in thousands of field multiplications m) per signature for BN-IBV, EdDSA, and ECDSA*. Batch verification to single verification ratios are included. The symbol (ted) indicates using twisted Edwards curves/formulas. 128-bit security is assumed throughout.

Chapter 5

Conclusions and Future Work

We find that in the context of batch verification, pairing-based signature schemes are not exceedingly inefficient when compared with non-pairing candidates. Specifically, the CHP/BN-IBV scheme is a leading candidate for V2V communication because of its identity privacy preservation property, in addition to having short signatures and keys. The cost of batch verification in BN-IBV is close to that of EdDSA using Bos-Coster, although further accounting must be made for speed of base field operations.

One area for future work would be to implement the Cheon-Yi batch verification algorithm on top of NaCl, the cryptographic library which implements Curve25519 and the EdDSA signature scheme. Based on the first order analysis in this thesis, the efficiency of Cheon-Yi and Bos-Coster batching must be evaluated on a finer-grained scale to get a clearer idea on which algorithm is most competitive in practice and adjust for differences in implementation, e.g. base field arithmetic. Simulating the number of group operations as we did for Bos-Coster in the case of Cheon-Yi would also help sharpen the comparison.

A security model and reductionist security proof for BN-IBV is needed. Another potential improvement on this work would come from refining the analysis to include modular additions, subtractions, and reductions. This would provide more fine-grained analysis but would require testing with software and implementation of the CHP signature scheme.

There are other elements that could be added to our discussion, as new work has been done on secure batching of ECDSA* signatures [73] and EdDSA signatures [72]. In addition, vehicle-to-vehicle communication has seen a number of advancements in security requirements [95, 98] and protocols [120]. The work in this thesis covered one important aspect of this discussion, however future work can focus on evaluating efficiency and implementation of schemes with additional properties that enhance the capabilities of vehicular

safety communication networks, see e.g. the V2V presentation from CHES 2014 [117].

References

[1] Car 2 Car. Communication consortium. http://car-to-car.org. 51

[2] iOS Security. https://ssl.apple.com/privacy/docs/iOS_Security_Guide_Oct_2014.pdf. 4

[3] NaCl: Networking and cryptography library. http://nacl.cr.yp.to/. 40

[4] Safecurves: Rho. http://safecurves.cr.yp.to/rho.html. 88

[5] SeVeCom. Security on the road. http://www.sevecom.org. 51

[6] Things that use curve25519. http://ianix.com/pub/curve25519-deployment.html. 4

[7] Trustpoint ECC Research. http://www.trustpointinnovation.com/ecc/. 4, 89

[8] IEEE standard for WAVE security services for applications and management messages. *IEEE Std 1609.2-2013*, pages 1–289, April 2013. 4, 51

[9] National Institute for Standards and Technology Digital Signature Standard. *FIPS Publication 186-4*, 2013. 7

[10] Joseph A. Akinyele, Matthew Green, Susan Hohenberger, and Matthew W. Pagano. Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 474–487, New York, NY, USA, 2012. ACM. 51

[11] Adrian Antipa, Daniel Brown, Robert Gallant, Rob Lambert, René Struik, and Scott Vanstone. Accelerated verification of ECDSA signatures. In *Selected Areas in Cryptography–SAC 2005*, pages 307–318. Springer, 2006. 8, 9, 21

[12] Diego F Aranha, Paulo SLM Barreto, Patrick Longa, and Jefferson E Ricardini. The realm of the pairings. In *Selected Areas in Cryptography–SAC 2013*, pages 3–25. Springer, 2014. 74

[13] Diego F Aranha, Koray Karabina, Patrick Longa, Catherine H Gebotys, and Julio López. Faster explicit formulas for computing pairings over ordinary curves. In *Advances in Cryptology–EUROCRYPT 2011*, pages 48–68. Springer, 2011. 62, 63, 64, 65, 67, 69, 71, 73, 74, 90

[14] Kenneth C Barr and Krste Asanović. Energy-aware lossless data compression. *ACM Transactions on Computer Systems (TOCS)*, 24(3):250–291, 2006. 51

[15] Paulo SLM Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In *Security in Communication Networks–SCN 2003*, pages 257–267. Springer, 2003. 61

[16] Paulo SLM Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography–SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer Berlin Heidelberg, 2006. 61

[17] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In *Advances in Cryptology–EUROCRYPT 2004*, pages 171–188. Springer, 2004. 32

[18] Mihir Bellare, Juan A Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology–EUROCRYPT'98*, pages 236–250. Springer, 1998. 2, 3, 4, 6, 16, 17

[19] Daniel J Bernstein. Pippenger's exponentiation algorithm. `cr.yp.to/papers/pippenger.pdf`, 2002.

[20] Daniel J Bernstein. Curve25519: new Diffie-Hellman speed records. In *Public Key Cryptography–PKC 2006*, pages 207–228. Springer, 2006. 26, 41

[21] Daniel J Bernstein. How to design an elliptic-curve signature system. `http://blog.cr.yp.to/20140323-ecdsa.html`, March 2014. 40

[22] Daniel J Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted Edwards curves. In *Progress in Cryptology–AFRICACRYPT 2008*, pages 389–405. Springer, 2008. ix, 26, 33, 34, 35, 36, 37, 42, 43

[23] Daniel J Bernstein, Johannes Buchmann, and Erik Dahmen. *Post-Quantum Cryptography*. Springer, 2009. 1

[24] Daniel J Bernstein, Jeroen Doumen, Tanja Lange, and Jan-Jaap Oosterwijk. Faster batch forgery identification. In *Progress in Cryptology–INDOCRYPT 2012*, pages 454–473. Springer, 2012. 3, 4, 6, 15, 47, 48

[25] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012. 6, 26, 37, 39, 44, 46, 50, 83, 85, 86, 90

[26] Daniel J Bernstein and Tanja Lange. Faster addition and doubling on elliptic curves. In *Advances in Cryptology–ASIACRYPT 2007*, pages 29–50. Springer, 2007. 33, 43

[27] Daniel J Bernstein and Tanja Lange. Inverted Edwards coordinates. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 20–27. Springer, 2007. 43

[28] Dan Boneh et al. Twenty years of attacks on the rsa cryptosystem. *Notices of the AMS*, 46(2):203–213. 1

[29] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology–CRYPTO 2001*, pages 213–229. Springer, 2001. 74

[30] Dan Boneh, Antoine Joux, and PhongQ. Nguyen. Why textbook elgamal and rsa encryption are insecure. In Tatsuaki Okamoto, editor, *Advances in Cryptology ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 30–43. Springer Berlin Heidelberg, 2000. 1

[31] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004. 52, 74, 75, 76

[32] Joppe W Bos, Craig Costello, Patrick Longa, and Michael Naehrig. Selecting elliptic curves for cryptography: An efficiency and security analysis. Technical Report MSR-TR-2014-19, Microsoft Research, 2014. 33

[33] Jurjen Bos and Matthijs Coster. Addition chain heuristics. In *Advances in Cryptology–CRYPTO'89*, pages 400–407. Springer, 1990. 45

[34] Wieb Bosma and Hendrik W Lenstra. Complete systems of two addition laws for elliptic curves. *Journal of Number Theory*, 53(2):229–240, 1995. 33

[35] Colin Boyd and Chris Pavlovski. Attacking and repairing batch verification schemes. In *Advances in Cryptology–ASIACRYPT 2000*, pages 58–71. Springer, 2000. 17

[36] Eric Brier and Marc Joye. Fast point multiplication on elliptic curves through isogenies. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 43–50. Springer, 2003. 14

[37] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. In *Advances in Cryptology–EUROCRYPT 2007*, pages 246–263. Springer, 2007. 51, 76, 81

[38] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology–CRYPTO 2004*, pages 56–72. Springer, 2004. 76

[39] Sanjit Chatterjee, Darrel Hankerson, Edward Knapp, and Alfred Menezes. Comparing two pairing-based aggregate signature schemes. *Designs, Codes and Cryptography*, 55(2-3):141–167, 2010. 52, 65

[40] Sanjit Chatterjee, Darrel Hankerson, and Alfred Menezes. On the efficiency and security of pairing-based protocols in the type 1 and type 4 settings. In *Arithmetic of Finite Fields–WAIFI 2010*, pages 114–134. Springer, 2010. 57

[41] Jung Hee Cheon and Dong Hoon Lee. Use of sparse and/or complex exponents in batch verification of exponentiations. *IEEE Transactions on Computers*, 55(12):1536–1542, 2006. 17

[42] Jung Hee Cheon and Jeong Hyun Yi. Fast batch verification of multiple signatures. In *Public Key Cryptography–PKC 2007*, pages 442–457. Springer, 2007. 17, 18

[43] Jaewook Chung and M Anwar Hasan. Asymmetric squaring formulae. In *18th IEEE Symposium on Computer Arithmetic, 2007. ARITH'07*, pages 113–122. IEEE, 2007. 66

[44] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. CRC Press, 2005. 36

[45] Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *Advances in Cryptology–ASIACRYPT'98*, pages 51–65. Springer, 1998. 41

[46] Don Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Transactions on Information Theory*, 30(4):587–594, 1984. 33

[47] Peter De Rooij. Efficient exponentiation using precomputation and vector addition chains. In *Advances in Cryptology–EUROCRYPT'94*, pages 389–399. Springer, 1995. 45, 46

[48] Augusto Jun Devegili, Michael Scott, and Ricardo Dahab. Implementing cryptographic pairings over barreto-naehrig curves. In *Pairing-Based Cryptography–Pairing 2007*, pages 197–207. Springer, 2007. 71, 72, 73

[49] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, Nov 1976. 1

[50] Harold Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44(3):393–422, 2007. 26, 33

[51] Torsten Ekedahl. *One semester of elliptic curves*. European Mathematical Society, 2006. 36

[52] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. Practical short signature batch verification. In *Topics in Cryptology–CT-RSA 2009*, pages 309–324. Springer, 2009. 51

[53] Amos Fiat. Batch RSA. In *Advances in Cryptology–CRYPTO'89*, pages 175–185. Springer, 1990. 2, 6

[54] Pierre-Alain Fouque and Mehdi Tibouchi. Indifferentiable hashing to Barreto–Naehrig curves. In *Progress in Cryptology–LATINCRYPT 2012*, pages 1–17. Springer, 2012. 82, 84

[55] Gerhard Frey, Michael Muller, and Hans-Georg Rück. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Transactions on Information Theory*, 45(5):1717–1719, 1999. 61

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

[57] Steven D Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012. 28

[58] Robert P Gallant, Robert J Lambert, and Scott A Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In *Advances in Cryptology–CRYPTO 2001*, pages 190–200. Springer, 2001. 11

[59] Eu-Jin Goh and Stanisław Jarecki. A signature scheme as secure as the Diffie-Hellman problem. In *Advances in Cryptology–EUROCRYPT 2003*, pages 401–415. Springer, 2003. 31

[60] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*, volume 2. Cambridge University Press, 2009. 27

[61] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[62] Robert Granger and Michael Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. In *Public Key Cryptography–PKC 2010*, pages 209–223. Springer, 2010. 71

[63] Keisuke Hakuta, Yosuke Katoh, Hisayoshi Sato, and Tsuyoshi Takagi. Batch verification suitable for efficiently verifying a limited number of signatures. In *Information Security and Cryptology–ICISC 2012*, pages 425–440. Springer, 2013. 21

[64] Darrel Hankerson, Scott Vanstone, and Alfred J Menezes. *Guide to Elliptic Curve Cryptography*. Springer, 2004. ix, 7, 10, 12, 13, 14, 15, 19, 35, 67, 85

[65] Robin Hartshorne. *Algebraic Geometry*. Springer, 1977. 35

[66] Ryan Henry and Ian Goldberg. Batch proofs of partial knowledge. In *Applied Cryptography and Network Security*, pages 502–517. Springer, 2013. 4

[67] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted Edwards curves revisited. In *Advances in Cryptology–ASIACRYPT 2008*, pages 326–343. Springer, 2008. 26, 44, 45, 86

[68] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001. 6, 7

[69] Koray Karabina. Squaring in cyclotomic subgroups. *Mathematics of Computation*, 82(281):555–579, 2013. 71

[70] Sabyasachi Karati and Abhijit Das. Using randomizers for batch verification of ECDSA signatures. *Cryptology ePrint Archive Report 2012/582*, 2012. 16

[71] Sabyasachi Karati and Abhijit Das. Batch Verification of EdDSA Signatures. In Rajat Subhra Chakraborty, Vashek Matyas, and Patrick Schaumont, editors, *Security, Privacy, and Applied Cryptography Engineering*, volume 8804 of *Lecture Notes in Computer Science*, pages 256–271. Springer International Publishing, 2014. 6

[72] Sabyasachi Karati and Abhijit Das. Batch verification of EdDSA signatures. In *Security, Privacy, and Applied Cryptography Engineering*, pages 256–271. Springer, 2014. 92

[73] Sabyasachi Karati and Abhijit Das. Faster batch verification of standard ECDSA signatures using summation polynomials. In *Applied Cryptography and Network Security–ACNS 2014*, pages 438–456. Springer, 2014. 92

[74] Sabyasachi Karati, Abhijit Das, Dipanwita Roychowdhury, Bhargav Bellur, Debojyoti Bhattacharya, and Aravind Iyer. Batch verification of ECDSA signatures. In *Progress in Cryptology–AFRICACRYPT 2012*, pages 1–18. Springer, 2012. 6, 15

[75] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 155–164. ACM, 2003. 31

[76] Bernd C Kellner. The Bernoulli number page. http://bernoulli.org. 49

[77] Neal Koblitz and Alfred Menezes. Another look at security definitions. *Advances in Mathematics of Communications*, 7(1):1–38, 2013. 31

[78] Neal Koblitz and Alfred J Menezes. Another look at "provable security". *Journal of Cryptology*, 20(1):3–37, 2007. 27, 28, 29, 32

[79] Fabian Kuhn and René Struik. Random walks revisited: Extensions of Pollards rho algorithm for computing multiple discrete logarithms. In *Selected Areas in Cryptography–SAC 2001*, pages 212–229. Springer, 2001.

[80] Tanja Lange. Algebraic geometry in cryptology. http://www.hyperelliptic.org/tanja/teaching/AGCrypto10/. 32

[81] Laurie Law and Brian J Matt. Finding invalid signatures in pairing-based batches. In *Cryptography and Coding*, pages 34–53. Springer, 2007. 51, 85

[82] Patrick Longa. Selecting elliptic curves for cryptography. http://patricklonga.webs.com/Presentation_CFRG_Selecting_Elliptic_Curves_for_Cryptography.pdf. 33

[83] Anna Lysyanskaya, Ronald L Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In *Selected Areas in Cryptography–SAC 1999*, pages 184–199. Springer, 2000. 76

[84] Brian J Matt. Identification of multiple invalid signatures in pairing-based batched signatures. In *Public Key Cryptography–PKC 2009*, pages 337–356. Springer, 2009. 51, 85

[85] Brian J Matt. Identification of multiple invalid pairing-based signatures in constrained batches. In *Pairing-Based Cryptography–Pairing 2010*, pages 78–95. Springer, 2010. 51

[86] Alfred Menezes. An introduction to pairing-based cryptography. In *Recent Trends in Cryptography*, pages 47–65, 2009.

[87] Victor Miller. Short programs for functions on curves. 1986. 55

[88] Victor S Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004. 55

[89] Bodo Möller. Fractional windows revisited: Improved signed-digit representations for efficient exponentiation. In *Information Security and Cryptology–ICISC 2004*, pages 137–153. Springer, 2005.

[90] Bodo Möller and Andy Rupp. Faster multi-exponentiation through caching: accelerating ECDSA signature verification. In *Security and Cryptography for Networks–SCN 2008*, pages 39–56. Springer, 2008. 21, 23, 24

[91] Peter L Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987. 26, 34

[92] James Muir and Douglas Stinson. Minimality and other properties of the width-w nonadjacent form. *Mathematics of Computation*, 75(253):369–384, 2006.

[93] David Naccache, David M'Raïhi, Serge Vaudenay, and Dan Raphaeli. Can DSA be improved? Complexity trade-offs with the digital signature standard. In *Advances in Cryptology–EUROCRYPT'94*, pages 77–85. Springer, 1995. 6

[94] Phong Q Nguyen and Igor E Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Designs, Codes and Cryptography*, 30(2):201–217, 2003. 40

[95] Bryan Parno and Adrian Perrig. Challenges in securing vehicular networks. In *Workshop on hot topics in networks (HotNets-IV)*, pages 1–6, 2005. 92

[96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *Advances in Cryptology–EUROCRYPT'96*, pages 387–398. Springer, 1996. 27, 31

[97] Jothi Rangasamy, Douglas Stebila, Colin Boyd, and Juan González Nieto. An integrated approach to cryptographic mitigation of denial-of-service attacks. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 114–123. ACM, 2011. 4

[98] Maxim Raya and Jean-Pierre Hubaux. Securing vehicular ad hoc networks. *Journal of Computer Security*, 15(1):39–68, 2007. 4, 51, 89, 92

[99] Reza Rezaeian Farashahi, Dustin Moody, and Hongfeng Wu. Isomorphism classes of Edwards curves over finite fields. *Finite Fields and Their Applications*, 18(3):597–612, 2012. ix, 35, 36

[100] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. 1

[101] Eric Rowland. Sums of consecutive powers. `http://www.math.rutgers.edu/~erowland/sumsofpowers.html`. 49

[102] Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. Cryptosystems based on pairing. In *The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan*, pages 135–148, 2000. 74

[103] Ana Helena Sánchez and Francisco Rodríguez-Henríquez. Neon implementation of an attribute-based encryption scheme. In *Applied Cryptography and Network Security–ACNS 2013*, pages 322–338. Springer, 2013. 51, 63, 64, 65, 67, 71, 74, 81, 83

[104] René Schoof. Nonsingular plane cubic curves over finite fields. *Journal of Combinatorial Theory, Series A*, 46(2):183–211, 1987. 35

[105] Peter Schwabe. EdDSA signatures and ed25519. `cryptojedi.org/peter/data/nancy-20120320.pdf`, March 2012. 39

[106] Peter Schwabe. Scalar Multiplication Algorithms. https://www.cosic.esat.kuleuven.be/ecc2013/files/peter_schwabe_2.pdf, September 2012. 85

[107] Michael Scott, Naomi Benger, Manuel Charlemagne, Luis J Dominguez Perez, and Ezekiel J Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. In *Pairing-Based Cryptography–Pairing 2009*, pages 78–88. Springer, 2009. 71

[108] Igor R Shafarevich. Basic Algebraic Geometry 1: Varieties in Projective Space. 2013. 35

[109] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology–CRYPTO 1984*, pages 47–53. Springer, 1985. 73

[110] Joseph H Silverman. *The Arithmetic of Elliptic Curves*, volume 106. Springer, 2009. 35

[111] Nigel P Smart and Frederik Vercauteren. On computable isomorphisms in efficient asymmetric pairing-based systems. *Discrete Applied Mathematics*, 155(4):538–547, 2007. 52

[112] Jerome A Solinas. Low-weight binary representations for pairs of integers. Technical Report CORR 2001-48, Department of C&O, University of Waterloo, 2001. 11, 12

[113] Martijn Stam and Arjen K Lenstra. Efficient subgroup exponentiation in quadratic and sixth degree extensions. In *Cryptographic Hardware and Embedded Systems–CHES 2002*, pages 318–332. Springer, 2003. 66

[114] Douglas R Stinson. *Cryptography: Theory and Practice*. CRC Press, 2005. 1, 2, 26

[115] Frederik Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010. 55, 61, 62

[116] Lawrence C Washington. *Elliptic Curves: Number Theory and Cryptography*. CRC press, 2012. 53, 54, 57

[117] Andr Weimerskirch. V2V Communication Security: A Privacy Preserving Design for 300 million vehicles. http://www.chesworkshop.org/ches2014/presentations/CHES_2014_Invited.pdfl. 89, 93

[118] Kewei Yu. Optimal pairings on BN curves. M.Math thesis, University of Waterloo. 2011. 54, 58, 62, 70, 73

[119] Gregory M Zaverucha and Douglas R Stinson. Group testing and batch verification. In *Information Theoretic Security*, pages 140–157. Springer, 2010. 3, 81

[120] Chenxi Zhang, Rongxing Lu, Xiaodong Lin, Pin-Han Ho, and Xuemin Shen. An efficient identity-based batch verification scheme for vehicular sensor networks. In *INFOCOM 2008. The 27th Conference on Computer Communications*. IEEE, 2008. 5, 51, 76, 80, 81, 82, 90, 92