On the Efficiency and Security of Cryptographic Pairings

by

Edward Knapp

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2012

Abstract

Pairing-based cryptography has been employed to obtain several advantageous cryptographic protocols. In particular, there exist several identity-based variants of common cryptographic schemes. The computation of a single pairing is a comparatively expensive operation, since it often requires many operations in the underlying elliptic curve. In this thesis, we explore the efficient computation of pairings.

Computation of the Tate pairing is done in two steps. First, a Miller function is computed, followed by the final exponentiation. We discuss the state-of-the-art optimizations for Miller function computation under various conditions. We are able to shave off a fixed number of operations in the final exponentiation. We consider methods to effectively parallelize the computation of pairings in a multi-core setting and discover that the Weil pairing may provide some advantage under certain conditions. This work is extended to the 192-bit security level and some unlikely candidate curves for such a setting are discovered.

Electronic Toll Pricing (ETP) aims to improve road tolling by collecting toll fares electronically and without the need to slow down vehicles. In most ETP schemes, drivers are charged periodically based on the locations, times, distances or durations travelled. Many ETP schemes are currently deployed and although these systems are efficient, they require a great deal of knowledge regarding driving habits in order to operate correctly. We present an ETP scheme where pairing-based BLS signatures play an important role.

Finally, we discuss the security of pairings in the presence of an efficient algorithm to invert the pairing. We generalize previous results to the setting of asymmetric pairings as well as give a simplified proof in the symmetric setting.

Acknowledgements

I would like to thank my supervisor Alfred Menezes for his help and support throughout my thesis work. I wish to thank my thesis committee for their comments and suggestions.

Table of Contents

CHAPTER 1

Introduction to Pairings

Pairing-based cryptography has been employed to obtain several advantageous cryptographic protocols. In particular, there exist several identity-based variants of common cryptographic schemes. Pairings can be roughly divided into two classes, based on the underlying elliptic curve. A pairing such that $e\colon G \times G \to G_T$ where G, G_T are prime order-r groups is known as a symmetric pairing (also called a *Type 1* pairing in the literature [36]). If $e\colon G_1 \times G_2 \to G_T$, where G_1, G_2, G_T are prime order-r groups and no efficient isomorphism between G_1 and G_2 is known, then e is called an asymmetric pairing. In the literature [36], asymmetric pairings are referred to as *Type 2* pairings when a one-way isomorphism $G_2 \to G_1$ is known and *Type 3* pairings if no such isomorphism is known. We consider only Type 3 pairings in this thesis. The groups G, G_1, and G_2 are constructed from elliptic curves and G_T is a multiplicative subgroup of a finite field. Care needs to be taken in choosing a pairing so that r is large enough to deflect Pollard's rho attack on the discrete logarithm problem in G and G_1 [71] and so that G_T is large enough to protect against the number field sieve attack on the discrete logarithm problem in G_T [77]. Symmetric pairings are constructed from *supersingular* elliptic curves, whereas asymmetric pairings are constructed from *ordinary* elliptic curves.

The computation of a single pairing is a comparatively expensive operation, since it often requires many group operations in G and G_T (or G_1, G_2, and G_T in the case of ordinary curves). We discuss several pairing-based protocols in this chapter. The BLS signature scheme provides us with short signatures, compared with analogous RSA-based signatures and elliptic-curve-based Elgamal signatures. The Waters signature scheme also provides relatively short signatures but operates with fewer security assumptions. When compared with BLS, these assumptions come at the cost of many non-pairing computations, potentially more costly than the pairing-related computations. In Section 1.1, we present the BLS signature scheme and an identity-based encryption scheme using symmetric pairings. Frequently, we can translate protocols from the symmetric setting to the asymmetric setting, as we do with the BLS signature scheme in Section 1.2. Further protocols are presented in Section 1.3 – a signature scheme and a key agreement protocol which requires exactly one pairing computation by each user. These protocols highlight the need for the efficient computation of a single pairing. In addition, we discuss multi-signature and aggregate signature schemes and the pairing computations required in these protocols.

1.1. Symmetric pairings

Let G and G_T be order-r groups, where r is prime. We write G as an additive group and G_T as a multiplicative group.

A *symmetric pairing* is a function

$$e\colon G \times G \to G_T$$

such that e is linear in each coordinate. That is,

$$e(aP, bQ) = e(P, Q)^{ab}$$

for all integers a, b, where P and Q are generators of G. In addition, we assume that e is non-degenerate (i.e. $e(P, Q) \neq 1$) and that e can be efficiently computed.

1.1.1. Hard problems. There are a few relatively standard security assumptions in pairing-based cryptography. For symmetric pairings, we require that the groups G and G_T be such that it is computationally infeasible to solve *the discrete log problem*: given generators P, $Q \in G$, find an integer a such that $aP = Q$.

There are a few related problems which rely on the discrete log problem being computationally infeasible. The computational Diffie-Hellman problem (CDH) in G is as follows: given P, aP, $bP \in G$, find $abP \in G$. Clearly CDH can be solved efficiently if the discrete log problem can be solved and it is usually assumed the converse it true. The decisional Diffie-Hellman problem (DDH) is to determine if a proposed solution to a CDH instance is correct. For symmetric pairings, the DDH problem in G can be efficiently solved – namely, for a potential solution Q to the instance P, aP, bP, we can check if

$$e(aP, bP) \overset{?}{=} e(P, Q).$$

The *computational bilinear Diffie-Hellman problem* (CBDH) is as follows: given P, aP, bP, $cP \in G$, determine $e(P, P)^{abc}$. Similarly, there exists a corresponding decisional variant called the *decisional bilinear Diffie-Hellman problem* (DBDH).

1.1.2. Symmetric BLS. The Boneh-Lynn-Shacham (BLS) signature scheme [18] is a pairing-based protocol that is analogous to the RSA-based Full-Domain Hash (FDH) signature scheme [13]. The BLS signature scheme has several advantages over RSA-FDH. In particular, signing is more efficient for BLS and signatures are smaller.

The BLS signature scheme has three algorithms: `Generate`, `Sign`, and `Verify`. We fix a hash function $H\colon \{0, 1\}^* \to G$, a pairing $e\colon G \times G \to G_T$, and a generator P of G as public parameters.

`Generate` selects a positive integer $x < r$ uniformly at random and returns the private key x and public key $X = xP$.

`Sign` takes as input a message m and a private key x. The algorithm computes $Q = H(m)$ and returns the signature $\sigma = xQ$.

`Verify` takes as input a message m, a signature σ, and a public key X. The algorithm computes $Q = H(m)$ and returns 'valid' if $e(Q, X) = e(\sigma, P)$ and 'invalid' otherwise.

The BLS signature scheme is secure against existential forgeries under adaptive chosen-message attacks [18]. Roughly speaking, this attacker is assumed to have the ability to induce a signer to sign arbitrary messages of the forger's choosing and has the goal of producing a new signature on a message of the forger's choosing. This is the standard security model for signature schemes. The computational assumption is that the CDH problem in G is difficult to solve. Moreover, the reductionist security proof assumes that H is a random function (the 'random oracle' assumption).

1.1.3. Identity-based encryption. Traditional public-key encryption schemes require that a sender obtain an authentic copy of the receiver's public key. Typically, a third party is used to sign a certificate ensuring the authenticity of a user's public key. Identity-based encryption schemes differ in that the encryption public key is not specific to any particular recipient but combined with the recipient's identity to encrypt the message. A third party is required to issue private keys to a recipient based on an identity. For instance, an email address can serve as an identity.

In order to send a message, the identity of the recipient and the public key of the third party are required. Additionally, encrypted messages can be sent to a recipient *before* the recipient obtains the private key required to decrypt it.

The Boneh-Franklin identity-based encryption scheme [16] consists of four algorithms, `Generate`, `Extract`, `Encrypt`, and `Decrypt`. The generate algorithm is used by the third party who is designated to issue private keys, the extract algorithm issues private keys to recipients, and the other two are for encryption and decryption. We present a simplified version of the scheme, which is only secure against passive attacks. The scheme can be made secure against active attacks with certain modifications which are omitted for simplicity.

Fix a pairing $e\colon G \times G \to G_T$ and generator P of G. Let

$$H_1\colon \{0,1\}^* \to G$$
$$H_2\colon G_T \to \{0,1\}^n$$

be hash functions, where $\{0,1\}^n$ is the message space.

`Generate` selects a positive integer $x < r$ uniformly at random and computes $X = xP$. The master key (that is, the third-party's private key) is x, the public key is X.

`Extract` takes as input an identity $\mathbf{ID}$ and a master key x. The algorithm computes $R = H_1(\mathbf{ID})$ and returns the private key $S_{\mathbf{ID}} = xR$ which corresponds to the identity $\mathbf{ID}$.

`Encrypt` takes as input a message m and an identity $\mathbf{ID}$. The algorithm selects a positive integer $s < r$ uniformly at random and computes $R = H_1(\mathbf{ID})$ and $\gamma = e(R, X)^s$. The algorithm returns the ciphertext

$$c = \langle c_1, c_2 \rangle = \langle m \oplus H_2(\gamma), sP \rangle.$$

`Decrypt` takes as input a ciphertext $c = \langle c_1, c_2 \rangle$ and the private key $S_{\mathbf{ID}}$. The algorithm computes

$$\gamma = e(S_{\mathbf{ID}}, c_2)$$

and returns the message

$$m = H_2(\gamma) \oplus c_1.$$

The security of the scheme relies on the CBDH problem being difficult for the pairing e, and that H_1, H_2 are random functions.

1.2. Asymmetric pairings

A more general pairing is the asymmetric pairing. Let G_1, G_2, and G_T be order-r groups, where r is prime. We write G_1, G_2 as additive groups, and G_T as a multiplicative group. An *asymmetric pairing* is a function

$$e \colon G_1 \times G_2 \to G_T$$

such that e is linear in each coordinate. That is,

$$e(aP, bQ) = e(P, Q)^{ab}$$

for all integers a, b, where P and Q are generators of G_1 and G_2 respectively. In addition, we assume that e is non-degenerate (i.e. $e(P, Q) \neq 1$) and that e can be efficiently computed. If we can compute an isomorphism between G_1 and G_2, then we can use e as a symmetric pairing. If no efficiently computable isomorphism is known, then the pairing is strictly asymmetric.

1.2.1. Hard problems. For asymmetric pairings, we require that it be computationally infeasible to solve the discrete log problem in the groups G_1, G_2, and G_T.

There are hard problems in the symmetric setting which are analogous to the hard problems of the asymmetric setting. The computational Diffie-Hellman problem (CDH) for (G_1, G_2) is as follows: given P, aP, $bP \in G_1$ and Q, $bQ \in G_2$, find $abP \in G_1$. Once again, the CDH problem in (G_1, G_2) can be solved efficiently if the discrete log problem in G_1 or G_2 can be solved and it is usually assumed the converse it true.

The *computational bilinear Diffie-Hellman problem* (CBDH) is as follows: given P, aP, bP, $cP \in G_1$ and Q, bQ, $cQ \in G_2$, determine $e(P, Q)^{abc}$. Similarly, there exists a corresponding decisional variant called the *decisional bilinear Diffie-Hellman problem* (DBDH). This version of the problem is equivalent to the Type-2-pairing version where an isomorphism $Q \mapsto P$ is known, since we can compute bP, cP from bQ, cQ.

1.2.2. Asymmetric BLS. The BLS signature scheme has three algorithms: `Generate`, `Sign`, and `Verify`. We fix a hash function $H\colon \{0,1\}^* \to G_1$, a pairing $e\colon G_1 \times G_2 \to G_T$, and a generator Q of G_2 as public parameters.

`Generate` selects a positive integer $x < r$ uniformly at random and returns the private key x and public key $X = xQ$.

`Sign` takes as input a message m and a private key x. The algorithm computes $P = H(m)$ and returns the signature $\sigma = xP$.

`Verify` takes as input a message m, a signature σ, and a public key X. The algorithm computes $P = H(m)$ and returns 'valid' if $e(P, X) = e(\sigma, Q)$ and 'invalid' otherwise.

The main advantage of using BLS in the asymmetric setting is that we can choose from a wider variety of elliptic curves. Table 1.1 gives a comparison of key sizes for BLS and RSA-FDH. While verification is usually more expensive for BLS, the BLS scheme has the advantage of smaller signature sizes and faster signing.

Security level	Elliptic curve	BLS signature bit-length	RSA signature bit-length
128	BN [10]	256	3072
192	KSS-18 [47]	512	7680
256	BLS-24 [8]	640	15360

TABLE 1.1. A comparison of signature sizes (in bits) for BLS and RSA at various security levels.

The asymmetric BLS signature scheme is secure if the CDH problem is difficult for (G_1, G_2) and H is a random function.

1.2.3. Waters signatures (W1). The Waters signature scheme [90] provides relatively short signatures (compared to RSA-FDH) having fewer security assumptions than BLS. In particular, Waters signatures are at least twice as long as BLS signatures but still smaller than RSA signatures. Waters presented an identity-based encryption scheme whose reductionist security proof does not use random oracles. Since an identity-based encryption scheme can be used to construct a signature scheme, we obtain a signature scheme with a reductionist security proof that does not require random oracles.

Waters defined the scheme in the symmetric setting, however the scheme extends naturally to the asymmetric setting. The asymmetric setting gives us the option to swap G_1 and G_2. Since one of these groups has a larger representation (cf. Section 2.4), this allows us to make a trade-off between key size and signature size.

The Waters signature scheme has three algorithms: `Generate`, `Sign`, and `Verify`. We fix a pairing $e\colon G_1 \times G_2 \to G_T$, a generator P of G_1, a generator Q of G_2, and using a collision-resistant hash function, we assume all messages have bit-length s.

`Generate` defines a function $H\colon \{0,1\}^s \to G_1$ as

$$H(m) = U_0 + \sum_{i=1}^{s} m_i U_i,$$

where $U_i \in G_1$ are chosen uniformly at random (and fixed) and m_i is the i-th bit of the message m. Let $X \in G_1$ be selected uniformly at random. The public key is $\mathcal{X} = e(X, Q)$, U_0, U_1, …, U_s. The private key is X. Note that the elements U_0, …, U_s provide an equivalent description of H. Hence we can say that the algorithm returns the public key $\mathcal{X}$, H and the private key X.

`Sign` takes as input a message m, a private key X and public key $\mathcal{X}$, H. The algorithm computes $R = H(m)$, selects a positive integer $t < r$ uniformly at random, and returns the signature σ, where $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle = \langle X + tR, tP, tQ \rangle$.

`Verify` takes as input a message m, a signature $\sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$, and a public key $\mathcal{X}$, H. The algorithm computes $R = H(m)$ and returns 'valid' if

$$e(\sigma_1, Q) = \mathcal{X} \cdot e(R, \sigma_3) \qquad \text{and} \qquad e(\sigma_2, Q) = e(P, \sigma_3),$$

and 'invalid' otherwise.

Although the element σ_2 seems unnecessary, it is required to prove the security of the scheme [22]. The Waters signature scheme is secure, assuming the CDH problem is difficult for (G_1, G_2).

1.2.4. Small-signature Waters (W2). We fix the same assumptions and define our three algorithms: `Generate`, `Sign`, and `Verify`.

`Generate` defines a function $K\colon \{0,1\}^s \to \mathbb{Z}$ as

$$K(m) = c_0 + \sum_{i=1}^{s} c_i \cdot m_i,$$

where the $c_i < r$ are positive integers chosen uniformly at random and m_i is the i-th bit of the message m. We define two functions $H_1\colon \{0,1\}^s \to G_1$, $H_2\colon \{0,1\}^s \to G_2$ which are related to K. Set $U_i = c_i P$ and $V_i = c_i Q$. Then define

$$H_1(m) = U_0 + \sum_{i=1}^{s} m_i U_i,$$

$$H_2(m) = V_0 + \sum_{i=1}^{s} m_i V_i.$$

Note that $[K(m)]P = H_1(m)$ and $[K(m)]Q = H_2(m)$. Let $X \in G_1$ be selected uniformly at random. The public key is $\mathcal{X} = e(X, Q)$, U_0, U_1, …, U_s, V_0, V_1, …, V_s. The private key is X. Note that the elements U_0, …, U_s, V_0, …, V_s provide an equivalent description of H_1 and H_2. Hence we can say that the algorithm returns the public key $\mathcal{X}$, H_1, H_2 and the private key X.

`Sign` takes as input a message m, a private key X, and public key $\mathcal{X}$, H_1, H_2. The algorithm computes $R = H_1(m)$, selects a positive integer $t < r$ uniformly at random, and returns the signature σ, where $\sigma = \langle \sigma_1, \sigma_2 \rangle = \langle X + tR, tP \rangle$.

`Verify` takes as input a message m, a signature $\sigma = \langle \sigma_1, \sigma_2 \rangle$, and a public key $\mathcal{X}$, H_1, H_2. The algorithm computes $R = H_2(m)$ and returns 'valid' if

$$e(\sigma_1, Q) = \mathcal{X} \cdot e(\sigma_2, R) \qquad \text{and} \qquad e(H_1(m), Q) = e(P, H_2(m)),$$

and 'invalid' otherwise.

The second verification condition, $e(H_1(m), Q) = e(P, H_2(m))$, can be guaranteed by ensuring that $e(U_i, Q) = e(P, V_i)$ for all i. Since this is independent of any message being verified, this condition can be ensured by precomputation or as a part of a public-key certification policy.

1.2.5. Comparison of the two Waters scheme. Table 1.2 gives a comparison of public key and signature sizes between the two variants of Waters for different security levels and corresponding parings.

We have not discussed computational considerations in this comparison. The significant number of group operations in G_1 and G_2 demanded by H, H_1, and H_2 will potentially contribute more than the cost of computing a pairing in the verification algorithm; for a detailed analysis, see Chatterjee *et al.*[22].

Security Level	Elliptic curve	W1 signature	W1 public key	W2 signature	W2 public key
128	BN [10]	1024	65729	512	197376
192	KSS-18 [47]	2560	197120	1024	788480
256	BLS-24 [8]	3840	328320	1280	1641600

Table 1.2. A comparison of the bit-lengths of public keys and signatures for the two Waters signature schemes.

1.3. More protocols

1.3.1. BB signatures. The Boneh-Boyen (BB) signature scheme [15] is a pairing-based signature scheme that has a reductionist security proof which does not use random oracles. We fix a pairing $e\colon G_1 \times G_2 \to G_T$ with generators P of G_1, Q of G_2, and define three algorithms: `Generate`, `Sign`, and `Verify`.

`Generate` selects positive integers $x < r$ and $y < r$ uniformly at random and sets $X_1 = xP$, $Y_1 = yP$, $X_2 = xQ$, $Y_2 = yQ$. The algorithm returns the private key $\langle x, y\rangle$ and the public key $\langle X_1, Y_1, X_2, Y_2\rangle$.

`Sign` takes as input a message $m \in [1, r-1]$ and a private key $\langle x, y\rangle$. The algorithm selects a positive integer $s < r$ uniformly at random such that $m + x + ys$ is not divisible by r. The signature is computed as $\sigma = \langle \sigma_1, \sigma_2\rangle = \langle (m + x + ys)^{-1}P, s\rangle$.

`Verify` takes as input a public key $\langle X_1, Y_1, X_2, Y_2\rangle$, a message m and a signature $\sigma = \langle \sigma_1, \sigma_2\rangle$. The algorithm returns 'valid' if

$$e(\sigma_1, mQ + X_2 + \sigma_2 Y_2) = e(P, Q),$$

and 'invalid' otherwise.

In order to verify a signature, the algorithm must ensure that $e(X_1, Q) = e(P, X_2)$ and $e(Y_1, Q) = e(P, Y_2)$. This can be guaranteed as part of the public-key certification procedure or precomputed by each verifier. Additionally, since $e(P, Q)$ is independent of the message being verified, the value $e(P, Q)$ can be precomputed and hence verification requires only a single pairing evaluation.

The BB signature scheme can securely sign s messages if the s-strong-Diffie-Hellman problem is infeasible: Given xQ, x^2Q, ..., x^sQ, xP, x^2P, ..., x^sP, determine $x^{s+1}P$. This version of the problem is equivalent to the Type-2-pairing version where an isomorphism $Q \mapsto P$ is known, since we can compute x^iP from x^iQ, for $i = 1, \ldots, s$.

1.3.2. Identity-based key agreement. Scott used pairings to design an identity-based key agreement scheme [78]. Traditional identity-based key agreement schemes require a two-way authenticated channel to form the shared secret. The identity-based scheme only requires that the parties be issued a private key by a trusted third party, based on their identity. The identity-based key agreement scheme consists of three algorithms `Generate`, `Extract`, and `Exchange`. We fix a pairing $e\colon G_1 \times G_2 \to G_T$ and hash functions

$$H_1\colon \{0,1\}^* \to G_1,$$
$$H_2\colon \{0,1\}^* \to G_2.$$

`Generate` selects a positive integer $x < r$ uniformly at random and returns the private key x.

`Extract` takes as input **ID** and computes $P = H_1(\mathbf{ID})$, $Q = H_2(\mathbf{ID})$ and returns the private key $S = xP$, $T = xQ$ which corresponds to the identity **ID**.

`Exchange` is an interactive algorithm. The algorithm requires two parties Alice and Bob with identities $\mathbf{ID}_A$ and $\mathbf{ID}_B$ respectively. Let $\langle S_A, T_A\rangle$ be the private key of Alice and

$\langle S_B, T_B \rangle$ the private key of Bob. The parties each select one of the groups G_1 and G_2. Without loss of generality, we say Alice has G_1 and Bob has G_2.

Alice		**Bob**
$\mathbf{ID}_A$	$\longrightarrow$	
	$\longleftarrow$	$\mathbf{ID}_B$
Choose positive integer $a < r$		Choose positive integer $b < r$
$B = H_2(\mathbf{ID}_B)$		$A = H_1(\mathbf{ID}_A)$
$\gamma_A = e(S_A, B)^a$	$\longrightarrow$	γ_A
γ_B	$\longleftarrow$	$\gamma_B = e(A, T_B)^b$
$\gamma = \gamma_B^a$		$\gamma = \gamma_A^b$

The result of the protocol is a session key, γ. The key agreement scheme is secure if the CBDH problem is computationally infeasible.

1.3.3. Multi-signatures and aggregate signatures. The BLS signature scheme (both symmetric and asymmetric) can be extended to multi-signature and aggregate signature schemes. A multi-signature scheme produces signatures (all by the same signer) which verify on a set of messages and which can be arbitrarily combined. In addition to the regular BLS algorithms, we obtain an additional algorithm `Combine` and a modified `Verify` algorithm. We present the modified schemes in terms of asymmetric pairings.

`Combine` takes as input two multi-signatures σ_1, σ_2 on disjoint sets of messages M_1, M_2 respectively. The algorithm returns the signature $\sigma = \sigma_1 + \sigma_2$ and the message set $M = M_1 \cup M_2$.

`Verify` takes as input a signature σ corresponding to the message set $M = \{m_1, \ldots, m_\ell\}$ and public key X. The algorithm sets $P_i = H(m_i)$ for each i. The algorithm returns 'valid' if

$$e(\sigma, Q) = e(\sum_{i=1}^{\ell} P_i, X),$$

and 'invalid' otherwise.

Boneh, Gentry, Lynn, and Shacham (BGLS) proposed an aggregate variant of the BLS signature scheme [17]. The BGLS signature scheme allows distinct messages signed by distinct signers to be combined into a single signature. We redefine the algorithms `Combine` and `Verify`.

`Combine` takes as input two aggregate signatures σ_1, σ_2 on disjoint sets of messages M_1, M_2 respectively signed by signers $\mathcal{X}_1$, $\mathcal{X}_2$ respectively. The algorithm returns the signature $\sigma = \sigma_1 + \sigma_2$, the message set $M = M_1 \cup M_2$, and the signers $\mathcal{X}_1 \cup \mathcal{X}_2$.

`Verify` takes as input a signature σ corresponding to the ordered message set $M = \{m_1, \ldots, m_\ell\}$ and ordered public key set $\mathcal{X} = \{X_1, \ldots, X_\ell\}$. The algorithm sets $P_i = H(m_i)$ for each i. The algorithm returns 'valid' if

$$e(\sigma, Q) = \prod_{i=1}^{\ell} e(P_i, X_i),$$

and 'invalid' otherwise.

1.4. BN pairings

In this section, we discuss a particular family of elliptic curves called Barreto-Naehrig (BN) curves [10]. This family of curves admit asymmetric pairings and are ideal for the 128-bit security level. BN curves have other nice features, such as a sextic twist, permitting a smaller representation for G_2 elements.

A BN elliptic curve

$$E : Y^2 = X^3 + b$$

of order r is defined over a prime field $\mathbb{F}_p$ where

$$p(z) = 36z^4 + 36z^3 + 24z^2 + 6z + 1,$$
$$r(z) = 36z^4 + 36z^3 + 18z^2 + 6z + 1,$$

are prime and $r = \#E(\mathbb{F}_p)$. These curves have embedding degree $k = 12$. The integer z is called the BN parameter.

Define $\pi\colon E \to E$, the p-th power Frobenius of E, by $\pi(x, y) = (x^p, y^p)$. Define the trace of E as $\mathrm{Tr}\colon E(\mathbb{F}_{p^k}) \to E(\mathbb{F}_p)$ by

$$\mathrm{Tr}(R) = \frac{1}{k} \sum_{i=0}^{k-1} \pi^i R.$$

The groups G_1 and G_2 are defined by $G_1 = E(\mathbb{F}_p)$ and

$$G_2 = \{Q \in E(\mathbb{F}_{p^k})[r] \mid \mathrm{Tr}\, Q = \infty\}.$$

The order-r Tate pairing (cf. Section 2.3.1) is computed as

$$G_1 \times G_2 \to G_T\colon (P, Q) \mapsto f_{r,P}(Q)^{(p^{12}-1)/r},$$

where $f_{r,P}$ is a Miller function requiring approximately $\log r$ operations to compute.

The optimal Tate pairing (cf. Section 3.1.2) is defined by

$$G_1 \times G_2 \to G_T\colon (P, Q) \mapsto \left(f_{6z+2,Q}(P) \cdot \ell_{(6z+2)Q,\pi(Q)}(P) \cdot \ell_{(6z+2)Q+\pi(Q),-\pi^2(Q)}(P)\right)^{(p^{12}-1)/r},$$

where $\ell_{A,B}$ denotes the line through points A and B, and where $f_{a,Q}$ denotes the rational function with a single zero at aQ, a pole with multiplicity a at Q, and a zero with multiplicity $a-1$ at ∞. This pairing is called "optimal" because the Miller function appearing in the optimal Tate pairing has roughly one-fourth the computational cost of the Miller function in the Tate pairing definition. The exponent $(p^{12}-1)/r$ in the Tate pairing computation can be written as $(p^6-1)(p^2+1)(p^4-p^2+1)/r$. Since p-th powering is inexpensive in $\mathbb{F}_{p^{12}}$, the exponentiation by $(p^6-1)(p^2+1)$ is said to be the *easy part* of the final exponentiation; the exponentiation by $(p^4-p^2+1)/r$ is called the *hard part*.

The Weil pairing (cf. Section 2.5.1) is defined as

$$G_1 \times G_2 \to G_T \colon (P,Q) \mapsto \frac{f_{r,P}(Q)}{f_{r,Q}(P)}.$$

Notice that the Weil pairing has two Miller functions and no final exponentiation. The computation of an optimal Tate pairing requires approximately $1/\varphi(k)$ times the computation of the Tate pairing. An analogous notion of optimality exists for the Weil pairing but with only a $1/2$ factor in savings. In particular

$$(P,Q) \mapsto \left(\frac{f_{2x,P}(Q)}{f_{2x,Q}(P)}\right)^{p^2+1} \left(\frac{f_{6x^2,P}(Q)\cdot \ell_{6x^2P,2xP}(Q)}{f_{6x^2,Q}(P)\cdot \ell_{6x^2Q,2xQ}(P)}\right)^{p^2} \frac{\ell_{2xP,P}(Q)}{\ell_{2xQ,Q}(P)}.$$

is a pairing (cf. Section 3.1.2). The functions $f_{2x,P}$, $f_{2x,Q}$ can be computed as intermediate steps in the computation of $f_{6x^2,P}$, $f_{6x^2,Q}$, hence the total computational cost is less than half of the Weil pairing.

1.5. Thesis outline

In most of this chapter, we have treated pairings as a black box, as is usually done when discussing protocols. In Chapter 2, we break open the box and discuss the construction and computation of pairings. We can divide all pairing constructions into two groups, the Tate pairing and the Weil pairing. All other pairings are known powers of the most basic Tate and Weil pairings. From any elliptic curve, we can construct either type of pairing but certain curves and certain protocols may make one more appealing than the other. In Chapter 2, we introduce a new method for proving that pairings are bilinear. Using this method, we present alternative proofs of bilinearity for variants of the Tate and Weil pairings. For example, instead of proving that the ate pairing is a power of the Tate pairing, we are able to prove bilinearity directly.

State-of-the-art pairing computation uses the product of powers of the Tate-like pairing, called optimal pairings. Vercauteren [87] developed a framework for efficient pairing computation using the product of powers of Tate pairings. Hess [42] extended this framework to the Weil pairing and also incorporated automorphisms to obtain a further computational advantage for certain curves. For curves where Hess is able to obtain a potential advantage using automorphisms, we show that the computation can be reduced even further. The optimal pairing framework is presented with an eye on implementation and we give example curves which illustrate the potential of each result.

For the BB signature scheme and Scott's identity-based key agreement scheme, a user requires the computation of a single pairing. Physical constraints on processor power has meant that computers frequently have many processors at their disposal. The expense of synchronizing these processors during a computation is relatively expensive compared to the cost of a pairing. Therefore, if we desire to compute a single pairing on, say, four processors, we may wish to give four independent computational instructions to the processors and combine the results when all have completed. Separating pairing computation into many independent computational tasks is examined in Chapter 4, where we discover that the Weil pairing is potentially faster than the Tate pairing under these conditions, contradicting conventional wisdom that the Tate pairing is superior to the Weil pairing.

The Hess-Vercauteren optimal pairing framework uses the structure of the elliptic curve to determine computationally advantageous pairings. In Chapter 5 we extend the framework to the final exponentiation, a large component of Tate pairing computation. We are able to reduce the computational expense by a fixed number of operations. Chapter 6 focuses on hashing to the group G_2. We determine the explicit action of automorphisms on elements of G_2 and combine this with inspiration from the optimal pairings framework to often double the efficiency of hashing to G_2.

At the 128-bit security level, BN curves are generally considered to be ideal. For higher security levels, the choice is not as obvious. Given that BN curves have so many appealing features, we might be tempted to use BN curves at the 192 bit security level as well. In Chapter 7, we compare the efficiency of pairings derived from several elliptic curves at the 192-bit security level. Surprisingly, we discover that an embedding degree 12 curve of composite order maintains many of the nice features of BN curves and outperforms other contenders that we consider.

Traditional stop-and-pay toll booths inconvenience drivers and are infeasible for complicated urban areas. As a way to minimize traffic congestion and avoid the inconveniences caused by toll booths, electronic tolling has been suggested. For example, as drivers pass certain locations, a picture of their licence plate may be taken and a bill sent to their home. However, this simplistic method allows the administrator of the system to build a dossier on drivers. While this may be an attractive feature for law enforcement, a society may not wish to trust the tolling agency with such detailed information. In Chapter 8, we present SPEcTRe, a suite of protocols utilizing the BLS or RSA-FDH signature scheme to maintain driver privacy while ensuring that tolls are accurately collected. While RSA-FDH offers extremely fast verification, BLS signatures have the advantage of being much smaller and having much more efficient signature generation.

For symmetric pairings $e : G \times G \rightarrow G_T$, Verheul proved that the existence of an efficiently-computable isomorphism $\phi : G_T \rightarrow G$ implies that the Diffie-Hellman problems in G and G_T can be efficiently solved [88]. In Chapter 9, we explore the implications of the existence of efficiently-computable isomorphisms $\phi_1 : G_T \rightarrow G_1$ and $\phi_2 : G_T \rightarrow G_2$

for asymmetric pairings $e : G_1 \times G_2 \to G_T$. We also give a simplified proof of Verheul's theorem.

CHAPTER 2

Background

At a high level, there are two flavours of pairings, the Tate pairings and the Weil pairings. All other pairings we discuss are known powers of these two pairings. The usual method to produce new pairings is to begin with the classic Tate or Weil pairing (which is parameterized by a large prime r) and show that known powers of the pairing can be computed differently. In this section, we provide direct proofs of bilinearity for several different Tate and Weil pairings.

We examine the structure of the Tate and Weil variants and place strict criteria on when they are bilinear. This permits us to give simple direct proofs of bilinearity for the classic Tate and Weil pairings and some of their variants.

Our focus is on proving that the pairings are bilinear. In addition, we require that the pairings are defined at all points and that the pairings are non-degenerate. Our method of proving bilinearity give no new insights into these issues. Thus, we defer to other works which address the matter. For the Tate pairing, we refer to the work of Hess [41]. For the Weil pairing, we refer to the work of Miller [61].

2.1. Properties of the Miller function

Let p be a prime power and let E be an elliptic curve over $\mathbb{F}_p$. Let r be a prime divisor of $\#E(\mathbb{F}_p)$ and suppose that $\gcd(p, r) = 1$. Let k be the least integer such that r divides $p^k - 1$, and suppose that $k \geq 2$. Then $E[r] \subseteq E(\mathbb{F}_{p^k})$ [5]. Assume further that r^2 does not divide $p^k - 1$. Let $\mathrm{Div}^0(E)$ denote the degree zero divisors over E.

Definition 2.1. For a group $H \subseteq E(\mathbb{F}_{p^k})$, let $\mathfrak{D}(H)$ denote the subset of divisors in $\mathrm{Div}^0(E)$ having support a subset of H.

Definition 2.2. For a point $P \in E$, define D_P as $(P) - (\infty)$.

Let $\pi\colon E \to E$ be the *p-th power Frobenius of E over* $\mathbb{F}_p$, defined by $\pi(x, y) = (x^p, y^p)$. Define $\mathrm{Tr}\colon E \to E$ to be the *trace map of E over* $\mathbb{F}_p$, which is computed as

$$\mathrm{Tr}\, P = \sum_{i=0}^{k-1} \pi^i P.$$

We define two special groups which are fixed throughout. Define $G_1 = E(\mathbb{F}_p)[r]$ and let G_2 be the Trace-0 subgroup of $E[r]$. That is, G_2 is the set of points $Q \in E[r]$ for which $\operatorname{Tr} Q = \infty$. We have $\pi Q = pQ$ for all $Q \in G_2$ [9, Lemma 3]. Let G_T denote the order-r multiplicative subgroup of $\mathbb{F}_{p^k}^*$.

Definition 2.3. For a function $\rho\colon H \to H$ and a divisor $D \in \mathfrak{D}(H)$ such that

$$D = \sum_{P \in H} n_P(P),$$

define

$$[\rho]D = \sum_{P \in H} n_P(\rho P).$$

In particular, for an integer m, we have

$$[m]D = \sum_{P \in H} n_P(mP),$$

where mP is the m-th multiple of P in the group H. This is in contrast to

$$mD = \sum_{P \in H} (m \cdot n_P)(P).$$

Let u_∞ be an $\mathbb{F}_p$-rational uniformizing parameter for ∞. A function $f \in \mathbb{F}_{p^k}(E)$ is said to be *normalized* if $lc_\infty(f) = 1$, where $lc_\infty(f) = (u_\infty^{-t} f)(\infty)$ and t is the order of f at ∞. Furthermore, f is said to be *semi-normalized* if $lc_\infty(f)$ belongs to a proper subfield of $\mathbb{F}_{p^k}$.

Definition 2.4. Let a be an integer and $D \in \operatorname{Div}^0(E)$. Define $f_{a,D}$ as the normalized rational function with divisor $[a]D - aD$, called the *Miller function*. The Miller function $f_{a,D}$ has length $|a|$ and bitlength $\log|a|$. Note that this is the definition of $f_{a,D}$ used by Hess [41], the inverse of other definitions for $f_{a,D}$.

Definition 2.5. Let a, b be integers and $D \in \operatorname{Div}^0(E)$. Define $g_{[a]D,[b]D}$ as the normalized rational function with divisor $[a+b]D - [a]D - [b]D$.

We give two interesting properties of Miller functions.

Lemma 2.1. Let a, b be integers and $D \in \operatorname{Div}^0(E)$. Then

$$f_{a+b,D} = f_{a,D} \cdot f_{b,D} \cdot g_{[a]D,[b]D}.$$

Lemma 2.2. Let a, b be integers and $D \in \operatorname{Div}^0(E)$. Then

$$f_{ab,D} = f_{b,D}^a \cdot f_{a,[b]D}.$$

Definition 2.6. Divisors D_1, $D_2 \in \operatorname{Div}^0(E)$ are *linearly equivalent*, denoted $D_1 \sim D_2$, if there exists a rational function with divisor $D_1 - D_2$. We denote the unique such normalized function by h_{D_1,D_2}.

Lemmas 2.3 and 2.4 give us a relation between Miller functions evaluated on linearly equivalent divisors.

Lemma 2.3. Let D_1 be linearly equivalent to D_2. Then

$$f_{a,D_1} = f_{a,D_2} \cdot \frac{h_{[a]D_1,[a]D_2}}{(h_{D_1,D_2})^a}.$$

Proof. Observe that

$$\operatorname{div} f_{a,D_1-D_2} = [a] \operatorname{div} h_{D_1,D_2} - a \operatorname{div} h_{D_1,D_2} = \operatorname{div} h_{[a]D_1,[a]D_2} - a \operatorname{div} h_{D_1,D_2}$$

and so

$$f_{a,D_1} = f_{a,D_2} \cdot f_{a,D_1-D_2} = f_{a,D_2} \cdot \frac{h_{[a]D_1,[a]D_2}}{(h_{D_1,D_2})^a}.$$

□

Lemma 2.4. Let D_1 be linearly equivalent to D_2. Let D have support such that $f_{a,D}$ is defined at D_1 and D_2. Then

$$f_{a,D}(D_1) = f_{a,D}(D_2) \cdot \frac{h_{D_1,D_2}([a]D)}{h_{D_1,D_2}(D)^a}.$$

Proof.

$$\begin{aligned} f_{a,D}(D_1) &= f_{a,D}(D_2) \cdot f_{a,D}(\operatorname{div} h_{D_1,D_2}) \\ &= f_{a,D}(D_2) \cdot h_{D_1,D_2}(\operatorname{div} f_{a,D}) \\ &= f_{a,D}(D_2) \cdot h_{D_1,D_2}([a]D - aD) \\ &= f_{a,D}(D_2) \cdot \frac{h_{D_1,D_2}([a]D)}{h_{D_1,D_2}(D)^a}, \end{aligned}$$

where the second equality is an application of Weil reciprocity [86, Excercise 2.11]. □

Lemmas 2.3 and 2.4 will be used to eventually prove bilinearity of certain functions. As an intermediate step, we will prove that certain functions satisfy the following definition.

Definition 2.7. Let $\mathfrak{D}_1$, $\mathfrak{D}_2$ be subgroups of $\operatorname{Div}^0(E)$. Let $e\colon \mathfrak{D}_1 \times \mathfrak{D}_2 \to \mathbb{F}_{p^k}^*$ be a function that may not be defined everywhere. We say e is *invariant under linear equivalence over* $\mathfrak{D}_1 \times \mathfrak{D}_2$, if for all divisors D_1, $D_1' \in \mathfrak{D}_1$, D_2, $D_2' \in \mathfrak{D}_2$, such that $D_1 \sim D_1'$, $D_2 \sim D_2'$ and $e(D_1, D_2)$ and $e(D_1', D_2')$ are defined, we have

$$e(D_1, D_2) = e(D_1', D_2').$$

When the domain is understood, we may simply say e is *invariant under linear equivalence.*

Definition 2.8. Let H_1, H_2 be subgroups of $E(\mathbb{F}_{p^k})$. Set $\mathfrak{D}_1 = \mathfrak{D}(H_1)$ and $\mathfrak{D}_2 = \mathfrak{D}(H_2)$. If $e\colon \mathfrak{D}_1 \times \mathfrak{D}_2 \to \mathbb{F}_{p^k}^*$ is invariant under linear equivalence, then define $e\colon H_1 \times H_2 \to \mathbb{F}_{p^k}^*$ as

$$e(P, Q) = e(D'_P, D'_Q),$$

where D'_P, D'_Q are linearly equivalent to D_P, D_Q and $e(D'_P, D'_Q)$ is defined.

All Miller functions are bilinear with respect to divisors. That is, for all integers a,

$$f_{a,D}(D_1 + D_2) = f_{a,D}(D_1) \cdot f_{a,D}(D_2),$$
$$f_{a,D_1+D_1}(D) = f_{a,D_1}(D) \cdot f_{a,D_2}(D),$$

where D, D_1, $D_2 \in \mathrm{Div}^0(E)$ and both sides of the equations are defined. For certain values of a, the Miller function is also bilinear over points. For example, we have

$$f_{a,D}(D_P + D_Q) = f_{a,D}(D_P) \cdot f_{a,D}(D_Q),$$

where $D \in \mathfrak{D}(G_2)$, P, $Q \in G_1$ and both sides of the equation are defined. In order to construct pairings over subgroups of the elliptic curve, we wish to construct functions that are bilinear over points. In the case that $a = p$, Lemma 2.10 gives us

$$f_{p,D}(D_{P+Q}) = f_{p,D}(D_P) \cdot f_{p,D}(D_Q).$$

That is, the function is bilinear with respect to points and not simply to divisors. We say that the function

$$e\colon \mathfrak{D}(G_2) \times \mathfrak{D}(G_1) \to \mathbb{F}_{p^k}^*$$

is bilinear, if it is bilinear with respect to divisors. On the other hand, we say the function

$$e\colon G_2 \times G_1 \to \mathbb{F}_{p^k}^*$$

is bilinear, if it is bilinear with respect to points. In other words, the domain of the function e determines whether we are talking about bilinearity with respect to divisors or bilinearity with respect to points.

The following lemma shows that we can generate a pairing that is bilinear with respect to points from maps that are bilinear with respect to divisors and invariant under linear equivalence.

Lemma 2.5. Let H_1, H_2 be subgroups of $E(\mathbb{F}_{p^k})$. Set $\mathfrak{D}_1 = \mathfrak{D}(H_1)$ and $\mathfrak{D}_2 = \mathfrak{D}(H_2)$. If $e\colon \mathfrak{D}_1 \times \mathfrak{D}_2 \to \mathbb{F}_{p^k}^*$ is bilinear (with respect to divisors) and invariant under linear equivalence, then $e\colon H_1 \times H_2 \to \mathbb{F}_{p^k}^*$ is also bilinear (with respect to points).

Proof. Let P, $Q \in H_1$. Let $R \in H_2$. Set $D_1 = D_P$, $D_2 = (P+Q) - (P)$, $D_3 = D_{P+Q}$. Notice that D_2 is linearly equivalent to D_Q and $D_1 + D_2 = D_3$. Let D be linearly equivalent to D_R with support disjoint to D_1, D_2, and D_3. Then

$$e(P+Q, R) = e(D_3, D) = e(D_1 + D_2, D) = e(D_1, D) \cdot e(D_2, D) = e(P, R) \cdot e(Q, R).$$

Symmetrically, the function $e\colon H_1 \times H_2 \to \mathbb{F}_{p^k}^*$ is a bilinear pairing. $\square$

For the remainder of the thesis, 'bilinearity' will simply refer to bilinearity with respect to points. Every application of Lemma 2.5 is to powers of Miller functions which, as noted above, are always bilinear with respect to divisors.

2.2. The Weil pairing

The most common instantiation of the Weil pairing is as the ratio of two order-r Miller functions. Miller proves bilinearity directly [61]; we rely on Lemma 2.5 to provide bilinearity by first proving invariance under linear equivalence. This method will give us intuition for producing other Weil pairings.

All variants of Weil pairings have the following general structure.

Definition 2.9. Let a be an integer. Let $D_1,\ D_2 \in \mathrm{Div}^0(E)$. Define w_a by

$$w_a(D_1, D_2) = \frac{f_{a,D_1}(D_2)}{f_{a,D_2}(D_1)}.$$

Note that w_a is not defined everywhere.

The following lemma describes the consequence of switching divisors in one coordinate.

Lemma 2.6. Let $D_1,\ D_1',\ D_2 \in \mathrm{Div}^0(E)$ be such that D_1' is linearly equivalent to D_1. Then

$$w_a(D_1, D_2) = w_a(D_1', D_2) \cdot \frac{h_{[a]D_1,[a]D_1'}(D_2)}{h_{D_1,D_1'}([a]D_2)}.$$

Proof. Applying Lemma 2.3 and Lemma 2.4, we have

$$\begin{aligned} w_a(D_1, D_2) &= \frac{f_{a,D_1}(D_2)}{f_{a,D_2}(D_1)} \\ &= \frac{f_{a,D_1'}(D_2)}{f_{a,D_2}(D_1')} \cdot \frac{h_{[a]D_1,[a]D_1'}(D_2)}{h_{D_1,D_1'}(D_2)^a} \cdot \frac{h_{D_1,D_1'}(D_2)^a}{h_{D_1,D_1'}([a]D_2)} \\ &= w_a(D_1', D_2) \cdot \frac{h_{[a]D_1,[a]D_1'}(D_2)}{h_{D_1,D_1'}([a]D_2)}. \end{aligned}$$

□

2.2.1. Order-r Weil pairing. Lemma 2.6 suggests that we could choose any a such that $h_{[a]D_1,[a]D_1'}(D_2) = h_{D_1,D_1'}([a]D_2)$ in order to obtain invariance under linear equivalence. Indeed, choosing $a = r$ will give us the desired result as demonstrated in the following lemma.

Lemma 2.7. The function w_r is invariant under linear equivalence over the domain $\mathfrak{D}(E[r]) \times \mathfrak{D}(E[r])$. Hence, the function

$$w_r \colon E[r] \times E[r] \to \mathbb{F}_{p^k}^*$$

is bilinear.

PROOF. Let D_1, D_1', $D_2 \in \mathfrak{D}(E[r])$ with $D_1' \sim D_1$. Applying Lemma 2.6, we obtain

$$\begin{aligned} w_r(D_1, D_2) &= w_r(D_1', D_2) \cdot \frac{h_{[r]D_1,[r]D_1'}(D_2)}{h_{D_1,D_1'}([r]D_2)} \\ &= w_r(D_1', D_2) \cdot \frac{h_{0,0}(D_2)}{h_{D_1,D_1'}(0)} \\ &= w_r(D_1', D_2). \end{aligned}$$

Since $w_r(D_1, D_2) = w_r(D_2, D_1)^{-1}$, by symmetry, the function $w_r \colon \mathfrak{D}(E[r]) \times \mathfrak{D}(E[r]) \to \mathbb{F}_{p^k}^*$ is invariant under linear equivalence. By Lemma 2.5, the function $w_r \colon E[r] \times E[r] \to \mathbb{F}_{p^k}^*$ is bilinear. □

2.2.2. Automorphism Weil pairing. The following result is a generalization of a result by Zhao *et al.* [91], extending their work to higher embedding degrees. Using Lemma 2.6 and the presence of an automorphism, we are able to construct a map that is invariant under linear equivalence. This pairing also supersedes an ate-like variation of the Weil pairing [92].

Lemma 2.8. Let ϕ be an automorphism of E and let λ be an integer such that $\phi P = \lambda P$ for all $P \in G_1$. The function w_λ is invariant under linear equivalence over the domain $\mathfrak{D}(G_1) \times \mathfrak{D}(G_2)$. Hence, the function

$$w_\lambda \colon G_1 \times G_2 \to \mathbb{F}_{p^k}^*$$

is bilinear.

PROOF. Let D_1, $D_1' \in \mathfrak{D}(G_1)$ with $D_1 \sim D_1'$ and let $D_2 \in \mathfrak{D}(G_2)$. Using Lemma 2.6, we have

$$\begin{aligned} w_\lambda(D_1, D_2) &= w_\lambda(D_1', D_2) \cdot \frac{h_{[\lambda]D_1,[\lambda]D_1'}(D_2)}{h_{D_1,D_1'}([\lambda]D_2)} \\ &= w_\lambda(D_1', D_2) \cdot \frac{h_{[\phi]D_1,[\phi]D_1'}(D_2)}{h_{D_1,D_1'}([\phi^{-1}]D_2)}, \end{aligned} \tag{2.1}$$

where $[\lambda]D_2 = [\phi^{-1}]D_2$ follows from the fact that $\phi Q = \lambda^{-1} Q$ for all $Q \in G_2$ [43, Section 6]. Since

$$h_{[\phi]D_1,[\phi]D_1'}(D_2) = (h_{D_1,D_1'} \circ \phi^{-1})(D_2) = h_{D_1,D_1'}([\phi^{-1}]D_2),$$

Equation (2.1) simplifies to

$$w_\lambda(D_1, D_2) = w_\lambda(D_1', D_2).$$

Symmetrically, we have for D_2' linearly equivalent to D_2 that

$$w_\lambda(D_1, D_2) = w_\lambda(D_1, D_2'),$$

proving the function $w_\lambda\colon \mathfrak{D}(G_1) \times \mathfrak{D}(G_2) \to \mathbb{F}_{p^k}^*$ is invariant under linear equivalence. By Lemma 2.5, the function $w_\lambda\colon G_1 \times G_2 \to \mathbb{F}_{p^k}^*$ is bilinear. □

For some elliptic curves, there exist automorphisms such that $\phi P = \lambda P$ and λ is approximately half the bitlength of r. In this case, w_λ can be computed twice as quickly as w_r.

2.3. The Tate pairing

The Tate pairing offers a few additional variants. All Tate pairings have the following general structure.

Definition 2.10. Let a be an integer. Let D_1, $D_2 \in \mathrm{Div}^0(E)$. Define τ_a by

$$\tau_a(D_1, D_2) = f_{a,D_1}(D_2).$$

Note that τ_a is not defined everywhere.

In addition, most variants require a *final exponentiation* to obtain a value that has order r in $\mathbb{F}_{p^k}^*$.

2.3.1. Order-r Tate pairing.

Lemma 2.9. The function $\tau_r^{(p^k-1)/r}$ is invariant under linear equivalence over the domain $\mathfrak{D}(E[r]) \times \mathfrak{D}(E(\mathbb{F}_{p^k}))$. Hence

$$\tau_r^{(p^k-1)/r}\colon E[r] \times E(\mathbb{F}_{p^k}) \to \mathbb{F}_{p^k}^*$$

is bilinear.

Proof. Let D_1, $D_1' \in \mathfrak{D}(E[r])$ be such that $D_1 \sim D_1'$. Let $D_2 \in \mathfrak{D}(E(\mathbb{F}_{p^k}))$ be such that $f_{r,D_1}(D_2)$ and $f_{r,D_1'}(D_2)$ are defined. Then

$$\begin{aligned} f_{r,D_1}(D_2)^{(p^k-1)/r} &= f_{r,D_1'}(D_2)^{(p^k-1)/r} \cdot \left(\frac{h_{[r]D_1,[r]D_1'}(D_2)}{h_{D_1,D_1'}(D_2)^r}\right)^{(p^k-1)/r} \\ &= f_{r,D_1'}(D_2)^{(p^k-1)/r} \cdot \frac{h_{[r]D_1,[r]D_1'}(D_2)^{(p^k-1)/r}}{h_{D_1,D_1'}(D_2)^{p^k-1}} \\ &= f_{r,D_1'}(D_2)^{(p^k-1)/r} \cdot h_{[r]D_1,[r]D_1'}(D_2)^{(p^k-1)/r}. \end{aligned}$$

Since D_1, D_1' have support in $E[r]$, we have that $h_{[r]D_1,[r]D_1'} = h_{0,0} = 1$ and so $f_{r,D_1}^{(p^k-1)/r} = f_{r,D_1'}^{(p^k-1)/r}$. This proves that we have invariance in the first coordinate.

For the second coordinate, fix $D_1 \in \mathfrak{D}(E[r])$. Let D_2, $D_2' \in \mathfrak{D}(E(\mathbb{F}_{p^k}))$ be such that $D_2 \sim D_2'$ and f_{r,D_1} is defined at D_2 and D_2'. Then

$$\begin{aligned} f_{r,D_1}(D_2)^{(p^k-1)/r} &= f_{r,D_1}(D_2')^{(p^k-1)/r} \cdot \left(\frac{h_{D_2,D_2'}([r]D_1)}{h_{D_2,D_2'}(D_1)^r} \right)^{(p^k-1)/r} \\ &= f_{r,D_1}(D_2')^{(p^k-1)/r} \cdot \frac{h_{D_2,D_2'}([r]D_1)^{(p^k-1)/r}}{h_{D_2,D_2'}(D_1)^{p^k-1}} \\ &= f_{r,D_1}(D_2')^{(p^k-1)/r} \cdot h_{D_2,D_2'}([r]D_1)^{(p^k-1)/r}. \end{aligned}$$

Since D_1 has support in $E[r]$, we have that $[r]D_1 = 0$ and so $f_{r,D_1}(D_2)^{(p^k-1)/r} = f_{r,D_1}(D_2')^{(p^k-1)/r}$, proving $\tau_r^{(p^k-1)/r} \colon \mathfrak{D}(E[r]) \times \mathfrak{D}(E(\mathbb{F}_{p^k}))$ is invariant under linear equivalence. By Lemma 2.5, the function $\tau_r^{(p^k-1)/r} \colon E[r] \times E(\mathbb{F}_{p^k}) \to \mathbb{F}_{p^k}^*$ is bilinear. $\square$

2.3.2. Order-p Tate pairing.

Lemma 2.10. [43] The function τ_p is invariant under linear equivalence over the domain $\mathfrak{D}(G_2) \times \mathfrak{D}(G_1)$. Hence

$$\tau_p \colon G_2 \times G_1 \to \mathbb{F}_{p^k}^*$$

is bilinear.

Proof. Let D_1, $D_1' \in \mathfrak{D}(G_2)$ be such that $D_1 \sim D_1'$. We have

$$\begin{aligned} f_{p,D_1} &= f_{p,D_1'} \cdot \frac{h_{[p]D_1,[p]D_1'}}{(h_{D_1,D_1'})^p} \\ &= f_{p,D_1'} \cdot \frac{h_{[\pi]D_1,[\pi]D_1'}}{(h_{D_1,D_1'})^p}. \end{aligned}$$

Let $D_2 \in \mathfrak{D}(G_1)$ be such that f_{p,D_1} and $f_{p,D_1'}$ are defined at D_2. We have that

$$(h_{D_1,D_1'})^p = h_{[\pi]D_1,[\pi]D_1'} \circ \pi$$

and since $[\pi]D_2 = D_2$, we obtain

$$\begin{aligned}
f_{p,D_1}(D_2) &= f_{p,D'_1}(D_2) \cdot \frac{h_{[\pi]D_1,[\pi]D'_1}(D_2)}{(h_{D_1,D'_1}(D_2))^p} \\
&= f_{p,D'_1}(D_2) \cdot \frac{h_{[\pi]D_1,[\pi]D'_1}(D_2)}{h_{[\pi]D_1,[\pi]D'_1}([\pi]D_2)} \\
&= f_{p,D'_1}(D_2) \cdot \frac{h_{[\pi]D_1,[\pi]D'_1}(D_2)}{h_{[\pi]D_1,[\pi]D'_1}(D_2)} \\
&= f_{p,D'_1}(D_2).
\end{aligned}$$

Next, let $D_1 \in \mathfrak{D}(G_2)$ and let D_2, $D'_2 \in \mathfrak{D}(G_1)$ be such that $D_2 \sim D'_2$ and f_{p,D_1} is defined at D_2 and D'_2. Then

$$\begin{aligned}
f_{p,D_1}(D_2) &= f_{p,D_1}(D'_2) \cdot \frac{h_{D_2,D'_2}([p]D_1)}{h_{D_2,D'_2}(D_1)^p} \\
&= f_{p,D_1}(D'_2) \cdot \frac{h_{D_2,D'_2}([\pi]D_1)}{h_{[\pi]D_2,[\pi]D'_2}([\pi]D_1)} \\
&= f_{p,D_1}(D'_2) \cdot \frac{h_{D_2,D'_2}([\pi]D_1)}{h_{D_2,D'_2}([\pi]D_1)} \\
&= f_{p,D_1}(D'_2).
\end{aligned}$$

Therefore $\tau_p \colon \mathfrak{D}(G_2) \times \mathfrak{D}(G_1) \to \mathbb{F}_{p^k}^*$ is invariant under linear equivalence. By Lemma 2.5, the function $\tau_p \colon G_2 \times G_1 \to \mathbb{F}_{p^k}^*$ is bilinear. □

2.3.3. Automorphism Tate pairing.

Lemma 2.11. [46] Let ϕ be an order-3 automorphism of E and let λ be an integer such that $\phi Q = \lambda Q$ for all $Q \in G_2$. Then the function

$$(D_1, D_2) \mapsto \left(\tau_\lambda(D_1, D_2)^{\lambda+1} \cdot \tau_{\lambda+1}([\phi]D_1, D_2)\right)^{(p^k-1)/r}$$

is invariant under linear equivalence over the domain $\mathfrak{D}(G_2) \times \mathfrak{D}(G_1)$ and the function

$$(D_1, D_2) \mapsto \left(\tau_\lambda(D_1, D_2)^{\lambda+1} \cdot \tau_{\lambda+1}([\phi^{-1}]D_1, D_2)\right)^{(p^k-1)/r}$$

is invariant under linear equivalence over the domain $\mathfrak{D}(G_1) \times \mathfrak{D}(G_2)$.

Proof. We have $mr = \Phi_3(\lambda) = \lambda^2 + \lambda + 1$ for some integer m, where Φ_s denotes the s-th cyclotomic polynomial. Therefore, for all $D \in \mathfrak{D}(E[r])$

$$f_{r,D}^m = f_{mr,D} = f_{\Phi_3(\lambda),D} = f_{\lambda^2+\lambda+1,D} = f_{\lambda^2+\lambda,D} = f_{(\lambda+1)\lambda,D} = f_{\lambda,D}^{\lambda+1} \cdot f_{\lambda+1,[\lambda]D}$$

and so

$$\tau_r(D_1, D_2)^m = \tau_\lambda(D_1, D_2)^{\lambda+1} \cdot \tau_{\lambda+1}([\lambda]D_1, D_2).$$

The value $[\lambda]D_1$ is either $[\phi]D_1$ or $[\phi^{-1}]D_1$ depending on the domain. Since $\tau_r^{(p^k-1)/r}$ is invariant under linear equivalence over the domains $\mathfrak{D}(G_2) \times \mathfrak{D}(G_1)$ and $\mathfrak{D}(G_1) \times \mathfrak{D}(G_2)$, the result holds. $\square$

The proof of the next result is similar to that of Lemma 2.11 and is omitted.

Lemma 2.12. [46] Let ϕ be an order-4 automorphism of E and let λ be an integer such that $\phi Q = \lambda Q$ for all $Q \in G_2$. Then the function

$$(D_1, D_2) \mapsto \left(\tau_\lambda(D_1, D_2)^\lambda \cdot \tau_\lambda([\phi]D_1, D_2)\right)^{(p^k-1)/r}$$

is invariant under linear equivalence over the domain $\mathfrak{D}(G_2) \times \mathfrak{D}(G_1)$ and the function

$$(D_1, D_2) \mapsto \left(\tau_\lambda(D_1, D_2)^\lambda \cdot \tau_\lambda([\phi^{-1}]D_1, D_2)\right)^{(p^k-1)/r}$$

is invariant under linear equivalence over the domain $\mathfrak{D}(G_1) \times \mathfrak{D}(G_2)$.

Lemma 2.13. [46] Let ϕ be an order-d automorphism such that d divides k and $\phi P = p^e P$ for all $P \in G_1$, where $e = k/d$. Then $\tau_{p^e}^{(p^k-1)/r}$ is invariant under linear equivalence over the domain $\mathfrak{D}(G_1) \times \mathfrak{D}(G_2)$.

Proof. Let $D_1 \in \mathfrak{D}(G_1)$ and $D_2 \in \mathfrak{D}(G_2)$. Let m be the integer such that $mr = p^k - 1$. Since $\phi P = p^e P$, we have $\phi Q = p^{-e} Q$ and so

$$\begin{aligned}
f_{r,D_1}(D_2)^m = f_{mr,D_1}(D_2) = f_{p^k-1,D_1}(D_2) = f_{p^k,D_1}(D_2) &= \prod_{i=0}^{d-1} f_{p^e,[p^{ei}]D_1}(D_2)^{p^{-ei-e}} \\
&= \prod_{i=0}^{d-1} f_{p^e,[\phi^i]D_1}(D_2)^{p^{-ei-e}} \\
&= \left(\prod_{i=0}^{d-1} f_{p^e,[\pi^{-ei}][\phi^i]D_1}([\pi^{-ei}]D_2)\right)^{p^{-e}} \\
&= \left(\prod_{i=0}^{d-1} f_{p^e,[\phi^i]D_1}([\pi^{-ei}]D_2)\right)^{p^{-e}} \\
&= \left(\prod_{i=0}^{d-1} f_{p^e,D_1}([\phi^{-i}][\pi^{-ei}]D_2)\right)^{p^{-e}} \\
&= \left(\prod_{i=0}^{d-1} f_{p^e,D_1}(D_2)\right)^{p^{-e}} \\
&= f_{p^e,D_1}(D_2)^{p^{-e}d}.
\end{aligned}$$

Since $\tau_r^m = \tau_r^{(p^k-1)/r}$ is invariant under linear equivalence, so is τ_{p^e}. $\square$

2.4. Twists

Definition 2.11. Let E, E' be elliptic curves defined over $\mathbb{F}_q$. If d is the least positive integer such that E and E are isomorphic over $\mathbb{F}_{q^d}$, then E' is an *order-d twist* of E.

Lemma 2.14. [43, Proposition 2] If there exists an order-d automorphism of E where d divides k, then there exists an order-d twist E' of E such that

$$E(\mathbb{F}_{p^k}) \cong E'(\mathbb{F}_{p^k})$$

and an isomorphism $\psi\colon E \to E'$ such that

$$\psi(x, y) = (\gamma^2 x, \gamma^3 y),$$

where $\gamma^d \in \mathbb{F}_{p^{k/d}}$.

Fix an order-d twist $\psi\colon E \to E'$. Set $G_1' = \psi G_1$, $G_2' = \psi G_2$, $q = p^{k/d}$, and $\pi' = \pi^{k/d}$. Next, we give a result which shows that the coordinates of G_2' lie in a smaller field.

Lemma 2.15. [43, Section 5] We can represent the group G_2' as

$$G_2' = E'(\mathbb{F}_{p^{k/d}})[r].$$

We have a limited number of potential automorphisms to choose from. The following lemma gives us that the automorphism group order is 1, 2, 3, 4, or 6 for the curves we will be considering.

Lemma 2.16. [86] If E is an ordinary elliptic curve and $\mathbb{F}_p$ has characteristic greater than 3, then the order of the automorphism group of E must divide 4 or 6.

Next, we observe an interesting property of G_1'.

Lemma 2.17. The points $P' \in G_1'$ satisfy $\operatorname{Tr}_{\pi'} P' = \infty$.

Proof. Let $P = (x, y) \in G_1$ and let $P' = \psi P$. Set $\phi = \psi^{-1} \circ \pi' \circ \psi$ and notice that

$$\phi(x, y) = (x\gamma^{2(q-1)}, y\gamma^{3(q-1)}).$$

Since $\gamma^d \in \mathbb{F}_q$, we have $\gamma^{3d(q-1)} = 1$ and so

$$\phi^d(x, y) = (x\gamma^{2d(q-1)}, y\gamma^{3d(q-1)}) = (x, y).$$

Therefore, ϕ is an automorphism of E with order dividing d and so $\phi P = q^s P$ for some integer s. Since r divides $q^d - 1 = (q-1)(1 + q + \cdots + q^{d-1})$ but not $q - 1$, we have that r divides $(1 + q + \cdots + q^{d-1})$ and so $(1 + q + \cdots + q^{d-1})P = \infty$. Since $\phi = \psi^{-1} \circ \pi' \circ \psi$, we have $\phi^i = \psi^{-1} \circ \pi'^i \circ \psi$. Rearranging, we obtain $\psi\phi^i = \pi'^i \circ \psi$, and so

$$\operatorname{Tr}_{\pi'} P' = \sum_{i=0}^{d-1} \pi'^i P' = \sum_{i=0}^{d-1} \pi^i \psi P = \sum_{i=0}^{d-1} \psi\phi^i P = \sum_{i=0}^{d-1} \psi q^{si} P = \psi\left(\sum_{i=0}^{d-1} q^{si} P\right).$$

Since q^s has order d modulo r, we have

$$\{q^{si} \mid i = 0, \dots, d-1\} = \{q^i \mid i = 0, \dots, d-1\}$$

and finally

$$\operatorname{Tr}_{\pi'} P' = \psi\left(\sum_{i=0}^{d-1} q^{si} P\right) = \psi\left(\sum_{i=0}^{d-1} q^{i} P\right) = \psi\infty = \infty.$$

□

Using twists, we can view E' over $\mathbb{F}_q$ as having embedding degree d (with respect to r). Since $G_2' = E'(\mathbb{F}_q)[r]$ and $G_1' \subseteq E'(\mathbb{F}_{q^d})[r]$ is such that $\operatorname{Tr}_{\pi'} P' = \infty$ for all $P' \in G_1'$, we can apply the Tate pairing results to E' as follows. Instead of applying the Tate pairing results to the tuple $(E, \mathbb{F}_p, k, G_1, G_2, \pi)$, we can substitute $(E', \mathbb{F}_q, d, G_2', G_1', \pi')$. In particular, the embedding degree is now d and the roles of G_1', G_2' are reversed. That is, G_1' acts analogously to G_2.

We apply the results from Section 2.3 to E' and the appropriate groups and use ψ to transfer the results back to E. Costello *et al.* [27] gave the following relation between Miller functions with divisors having support in E and Miller functions with divisors having support in E'.

Lemma 2.18. Let D_1, $D_2 \in \mathfrak{D}(E(\mathbb{F}_{p^k}))$ and set $D_1' = [\psi]D_1$, $D_2' = [\psi]D_2$. Then there exists a d-th root of unity ω such that

$$f_{a,D_1}(D_2) = \omega \cdot f_{a,D_1'}(D_2').$$

Lemma 2.18 immediately gives us twisted variants of Tate and Weil pairings. The order-r twisted Tate pairing is indistinguishable from the order-r Tate pairing. Similarly, the twisted Tate automorphism pairing does not give us a new domain. However, the q-order twisted pairing is of marginal interest.

Lemma 2.19. [27, Theorem 1] The function $\omega \cdot \tau_q$ is invariant under linear equivalence over the domain $\mathfrak{D}(G_1) \times \mathfrak{D}(G_2)$, where ω is a fixed d-th root of unity.

Weil-like functions can be proven to be bilinear using twists but no new Weil functions have been found.

2.5. Evaluation of Miller functions

So far, we have been looking at pairings evaluated at divisors. Now, we will look at the pairings evaluated at single points.

2.5.1. Weil pairings. We define a Weil pairing on points as follows.

Definition 2.12. Let H_1, H_2 be subgroups of $E(\mathbb{F}_{p^k})$. Define $w_a \colon H_1 \times H_2 \to \mathbb{F}_{p^k}^*$ by

$$w_a \colon (P, Q) \mapsto (-1)^a \frac{f_{a,D_P}(Q)}{f_{a,D_Q}(P)}.$$

Miller [61] showed that the Weil pairing $w_r(D_P, D_Q)$ can be computed by evaluating $w_r(P, Q)$.

Lemma 2.20. [61, Proposition 8] For all points P, $Q \in E[r]$, we have

$$w_r(D_P, D_Q) = w_r(P, Q).$$

Let ϕ be an automorphism of E, and let λ be an integer such that $\lambda Q = \phi Q$ for all $Q \in G_2$. To prove that we can evaluate w_λ at points, we give a relation between w_λ and w_r and invoke Lemma 2.20.

Lemma 2.21. Let ϕ be an order-b automorphism of E and let λ be an integer such that $\lambda Q = \phi Q$ for all $Q \in G_2$. Then for all $P \in G_1$, $Q \in G_2$, we have

$$w_r(D_P, D_Q)^{(\lambda^b - 1)/r} = w_\lambda(D_P, D_Q)^{b\lambda^{-1}}.$$

Proof. Let $D_1 \in \mathfrak{D}(G_1)$, $D_2 \in \mathfrak{D}(G_2)$ have disjoint support. We have

$$f_{r,D_1}^{(\lambda^b-1)/r} = f_{\lambda^b - 1, D_1} = f_{\lambda^b, D_1} = \prod_{i=0}^{b-1} f_{\lambda,[\phi^{-i}]D_1}^{\lambda^{-i-1}}$$

and symmetrically,

$$f_{r,D_2}^{(\lambda^b-1)/r} = \prod_{i=0}^{b-1} f_{\lambda,[\phi^{i}]D_2}^{\lambda^{-i-1}} = \prod_{i=0}^{b-1} f_{\lambda,D_2}^{\lambda^{-i-1}} \circ [\phi^{-i}].$$

We have

$$w_\lambda(D_1, D_2)^\lambda = w_\lambda([\lambda]D_1, D_2) = w_\lambda([\phi^{-1}]D_1, D_2)$$

and so

$$\begin{aligned} w_r(D_1, D_2)^{(\lambda^b-1)/r} &= \prod_{i=0}^{b-1} \left(\frac{f_{\lambda,[\phi^{-i}]D_1}(D_2)}{f_{\lambda,D_2}([\phi^{-i}]D_1)} \right)^{\lambda^{-i-1}} \\ &= \prod_{i=0}^{b-1} w_\lambda([\phi^{-i}]D_1, D_2)^{\lambda^{-i-1}} \\ &= \prod_{i=0}^{b-1} w_\lambda(D_1, D_2)^{\lambda^{-1}} \\ &= w_\lambda(D_1, D_2)^{b\lambda^{-1}}. \end{aligned}$$

□

Combining Lemma 2.20 with Lemma 2.21, we obtain the following.

Lemma 2.22. Let $P \in G_1$, $Q \in G_2$. Then

$$w_\lambda(D_P, D_Q) = w_\lambda(P, Q).$$

This allows us to evaluate the Weil pairings (Lemmas 2.7 and 2.8) at points instead of divisors. Next, we give similar results for the Tate pairing.

2.5.2. Tate pairings.

Definition 2.13. Let H_1, H_2 be subgroups of $E(\mathbb{F}_{p^k})$. Define $\tau_a \colon H_1 \times H_2 \to \mathbb{F}_{p^k}^*$ by

$$\tau_a \colon (P, Q) \mapsto f_{a,D_P}(Q).$$

From the proofs of the Tate pairing lemmas (Lemmas 2.9 and 2.10), we obtain a relationship between τ_a over different domains. In particular, we can evaluate f_{a,D_P} at points instead of divisors.

Lemma 2.23. Let D be linearly equivalent to D_P, let Q be distinct from P and ∞, and let a be an integer. Then since f_{a,D_P} is normalized, we have $f_{a,D_P}(\infty) = 1$ and so

$$f_{r,D}(D_Q)^{(p^k-1)/r} = \left(\frac{f_{r,D}(Q)}{f_{r,D}(\infty)}\right)^{(p^k-1)/r} = f_{r,D_P}(Q)^{(p^k-1)/r},$$

and

$$f_{p,D}(D_Q) = \frac{f_{p,D}(Q)}{f_{p,D}(\infty)} = f_{p,D_P}(Q).$$

Since we have derived a relationship between τ_r and the automorphism Tate pairings in the proof of Lemma 2.11, we can evaluate all Tate pairings at points instead of divisors, including twisted Tate pairings.

For $P \in E(\mathbb{F}_{p^k})$, define $f_{a,P} = f_{a,D_P}$. Using Lemma 2.1, the function $f_{a,P}(Q)$ can be computed with the following algorithm. Each iteration of the main loop of this algorithm is called a *Miller operation*.

To compute $f_{a,P}(Q)$, we give input $f = 1$, P, Q, a and ignore the aP result, keeping only $f_{a,P}(Q)$. The additional inputs and output of Miller's algorithm will be useful in future chapters and contribute no additional cost to the computation.

Algorithm 2.1 (Miller's Algorithm)

```
INPUT: $f$, $P$, $Q \in E(\mathbb{F}_{p^k})$, $a = \sum_{i=0}^{L} a_i 2^i$, where $a_i \in \{0,1\}$.
OUTPUT: $f^a \cdot f_{a,P}(Q)$, $aP$.
    1. $T \longleftarrow P$.
    2. For $i$ from $L$ down to 0 do:
        2.1 Let $\ell$ be the tangent line to $E$ at $T$.
        2.2 $T \longleftarrow 2T$.
        2.3 Let $v$ be the line through $T$ and $-T$.
        2.4 $f \longleftarrow f^2 \cdot \ell(Q)/v(Q)$.
        2.5 If $a_i = 1$ and $i \neq 0$ then
                Let $\ell$ be the line through $T$ and $P$.
                $T \longleftarrow T + P$,
                Let $v$ be the line through $T$ and $-T$.
                $f \longleftarrow f \cdot \ell(Q)/v(Q)$.
    3. Return $f$, $T$.
```

2.5.3. Denominator elimination. For elliptic curves E with even embedding degree k, there exists a twist E' of even order, d. Thus we have $G_1 = E(\mathbb{F}_p)[r]$ and $G_2 \cong E'(\mathbb{F}_{p^{k/d}})[r]$. Since $G_2 \subseteq E(\mathbb{F}_{p^k})$, performing G_2 operations on the twisted curve allows us to work over the smaller field $\mathbb{F}_{p^{k/d}}$, saving computation.

Recall that $\psi\colon E \to E'$ is computed as $\psi(x,y) = (\gamma^2 x, \gamma^3 y)$, where $\gamma \in \mathbb{F}_{p^k}$ is such that $\gamma^d \in \mathbb{F}_{p^{k/d}}$. Since d is even, we can take γ to be such that $\gamma^2 \in \mathbb{F}_{p^{d/2}}$. For a point $(x,y) \in G_2$, since $\gamma^2 x \in \mathbb{F}_{p^{k/d}}$, we have $x \in \mathbb{F}_{p^{k/d}}$. As first observed in [9], this property can be used to remove a significant portion of the cost of the Miller function.

The vertical line intersecting P, denoted v_P, is such that $v_P(x,y) = x - x_P$, where (x_P, y_P) are the coordinates of P. For $P \in G_1$, $Q \in G_2$, we have $v_P(Q) = x_Q - x_P$. Since $x_Q \in \mathbb{F}_{p^{k/d}}$ and $x_P \in \mathbb{F}_p$, we have that $v_P(Q)$ is in a proper subfield of $\mathbb{F}_{p^k}$. This yields the following result.

Lemma 2.24. For all $P \in G_1$, $Q \in G_2$, we have

$$v_P(Q)^{(p^k-1)/\Phi_k(p)} = 1,$$
$$v_Q(P)^{(p^k-1)/\Phi_k(p)} = 1.$$

Lemma 2.24 allows us to ignore all vertical lines $v_Q(P)$ in pairing computations where (P,Q) is in $G_1 \times G_2$ or $G_2 \times G_1$, so long as we exponentiate by $(p^k-1)/\Phi_k(p)$. Since r divides $p^k - 1$ and for all $m < k$, the value r does not divide $p^m - 1$, we have that r does not divide $(p^k-1)/\Phi_k(p)$. Hence exponentiation by $(p^k-1)/\Phi_k(p)$ maintains non-degeneracy. The cost of exponentiating by $(p^k-1)/\Phi_k(p)$ is relative low, given that $(p^k-1)/\Phi_k(p)$ expands into a polynomial in p with small coefficients and p-th powering is relatively cheap. This also turns out to be an intermediate step in computing the order-r Tate pairing, which has a final exponentiation of $(p^k - 1)/r$.

Miller's algorithm with denominator elimination is given as follows.

Algorithm 2.2 (Miller's Algorithm with denominator elimination)

```
INPUT: $f$, $P \in G_2$, $Q \in G_1$, $a = \sum_{i=0}^{L} a_i 2^i$, where $a_i \in \{0, 1\}$.
OUTPUT: $(f^a \cdot f_{a,P}(Q))^{(p^k-1)/\Phi_k(p)}$, $aP$.
    1. $T \longleftarrow P$.
    2. For $i$ from $L$ down to 0 do:
        2.1 Let $\ell$ be the tangent line to $E$ at $T$.
        2.2 $T \longleftarrow 2T$.
        2.3 $f \longleftarrow f^2 \cdot \ell(Q)$.
        2.4 If $a_i = 1$ and $i \neq 0$ then
                Let $\ell$ be the line through $T$ and $P$.
                $T \longleftarrow T + P$,
                $f \longleftarrow f \cdot \ell(Q)$.
    3. Return $f^{(p^k-1)/\Phi_k(p)}$, $T$.
```

2.6. Summary of results

In this section, we summarize the various pairing we have explored.

2.6.1. Weil pairings. Recall that G_T is the order-r multiplicative subgroup of $\mathbb{F}_{p^k}^*$. First, we observe that our Weil pairings produce elements in G_T.

Lemma 2.25. For all points P, $Q \in E[r]$, we have

$$w_r(P, Q) \in G_T.$$

PROOF. We must show that $w_r(P, Q)$ has order dividing r. We have

$$1 = w_r(0, D_Q) = w_r([r]D_P, D_Q) = w_r(D_P, D_Q)^r = w_r(P, Q),$$

proving that $w_r(P, Q) \in G_T$. □

Now, we give our two Weil pairings.

Theorem 2.26. The following is a pairing:

$$w_r \colon E[r] \times E[r] \to G_T \colon (P, Q) \mapsto \frac{f_{r,P}(Q)}{f_{r,Q}(P)}.$$

Theorem 2.27. Let λ be an integer such that $\phi Q = \lambda Q$ for all $Q \in G_2$, where ϕ is an automorphism of E. Then the following is a pairing:

$$G_1 \times G_2 \to G_T \colon (P, Q) \mapsto \frac{f_{\lambda,P}(Q)}{f_{\lambda,Q}(P)}.$$

The previous theorem can also be stated in the domain $G_2 \times G_1$ but this is only a superficial difference.

2.6.2. Tate pairings. Similarly, we show that our Tate pairings produce elements in G_T.

Lemma 2.28. For all points P, $Q \in E[r]$, we have

$$\tau_r(P, Q)^{(p^k-1)/r} \in G_T.$$

For all points $P \in G_2$, $Q \in G_1$, we have

$$\tau_p(P, Q) \in G_T.$$

Proof. Since $\tau_r(P, Q) \in \mathbb{F}_{p^k}^*$, we have $\tau_r(P, Q)^{(p^k-1)/r}$ has order dividing r. We have

$$1 = \tau_p(0, D_Q) = \tau_p([r]D_P, D_Q) = \tau_p(D_P, D_Q)^r = \tau_p(P, Q)^r,$$

proving that $\tau_p(P, Q) \in G_T$. □

Theorem 2.29. The following is a pairing:

$$E[r] \times E(\mathbb{F}_{p^k}) \to G_T \colon (P, Q) \mapsto f_{r,Q}(P)^{(p^k-1)/r}.$$

Theorem 2.30. For all integers i, the following is a pairing:

$$\tau_{p^i} \colon G_2 \times G_1 \to G_T \colon (P, Q) \mapsto f_{p^i,Q}(P).$$

Proof. By Lemma 2.10, we have $\tau_p \colon G_2 \times G_1 \to G_T$ is a pairing. Since

$$\begin{aligned}\tau_{p^i}(P, Q) &= f_{p^i,D_P}(Q) \\ &= \prod_{j=0}^{i-1} f_{p,[p^j]D_P}(Q)^{p^{i-1-j}} \\ &= \prod_{j=0}^{i-1} \tau_p(p^j P, Q)^{p^{i-1-j}}\end{aligned}$$

is the product of powers of τ_p pairings, the result follows. □

Theorem 2.31. Let 3 divide d. Let λ be an integer such that $\phi Q = \lambda Q$ for all $Q \in G_2$, where ϕ is an order-d automorphism of E. The following is a pairing:

$$G_2 \times G_1 \to G_T \colon (P, Q) \mapsto \left(f_{\lambda,Q}(P)^{\lambda+1} \cdot f_{\lambda+1,\phi Q}(P)\right)^{(p^k-1)/r}.$$

Theorem 2.32. Let 4 divide d. Let λ be an integer such that $\phi Q = \lambda Q$ for all $Q \in G_2$, where ϕ is an order-d automorphism of E. The following is a pairing:

$$G_2 \times G_1 \to G_T \colon (P, Q) \mapsto \left(f_{\lambda,Q}(P)^{\lambda} \cdot f_{\lambda,\phi Q}(P)\right)^{(p^k-1)/r}.$$

2.6.3. Twisted pairings.

Theorem 2.33. Set $q = p^{k/m}$, where m divides k and m is the order of some automorphism of E. Let i be an integer. The following is a pairing:

$$G_1 \times G_2 \to G_T \colon (P, Q) \mapsto \omega \cdot f_{q^i,Q}(P),$$

where ω is an m-th root of unity.

The following two theorems actually follow from the Tate pairing results in Section 2.3 but they fit better with twisted pairings, since they share a similar structure and domain.

Theorem 2.34. Let 3 divide d and let λ be such that $\phi Q = \lambda Q$ for all $Q \in G_2$, where ϕ is an order-d automorphism of E. The following is a pairing:

$$G_1 \times G_2 \to G_T \colon (P, Q) \mapsto \left(f_{\lambda,Q}(P)^{\lambda+1} \cdot f_{\lambda+1,\phi Q}(P)\right)^{(p^k-1)/r}.$$

Theorem 2.35. Let 4 divide d and let λ be such that $\phi Q = \lambda Q$ for all $Q \in G_2$, where ϕ is an order-d automorphism of E. The following is a pairing:

$$G_1 \times G_2 \to G_T \colon (P, Q) \mapsto \left(f_{\lambda,Q}(P)^{\lambda} \cdot f_{\lambda,\phi Q}(P)\right)^{(p^k-1)/r}.$$

CHAPTER 3

Hess-Vercauteren Optimal Pairings

The previous chapter was devoted to proving the bilinearity of the most basic Tate and Weil pairings. By considering products of powers of these pairings, we are able to obtain pairings which can be computed more efficiently. For instance, the ate pairing [43] can be computed more quickly than the order-r Tate pairing.

Lemma 3.1. Let $\lambda \equiv p^i \pmod{r}$. Then the map $\tau_\lambda^{(p^k-1)/r}\colon G_2 \times G_1 \to G_T$ is a pairing.

Proof. Let $D \in \mathfrak{D}(G_2)$. We have $\lambda = p^i + mr$ for some integer r. We have

$$\begin{aligned} f_{\lambda,D} &= f_{p^i+mr,D} \\ &= f_{p^i,D} \cdot f_{mr,D} \cdot g_{[p^i]D,[mr]D} \\ &= f_{p^i,D} \cdot f_{r,D}^m, \end{aligned}$$

where $g_{[p^i]D,[mr]D}$ has divisor $[p^i]D + [mr]D - [p^i + mr]D = 0$. Thus,

$$\tau_\lambda(P,Q) = \tau_{p^i}(P,Q) \cdot \tau_r(P,Q)^m.$$

Since $\tau_\lambda^{(p^k-1)/r}$ is the product of powers of the pairings τ_{p^i} and $\tau_r^{(p^k-1)/r}$, the result holds. $\square$

Since $p \equiv t - 1 \pmod{r}$, where t is the trace of the Frobenius acting on E and t is approximately half the bitlength of r, the computational cost of computing the Miller function is reduced by at least half. For all $\lambda \equiv p^i \pmod{r}$, we have that r divides $\Phi_m(\lambda)$ for some m dividing k. Since $\Phi_m(x)$ is irreducible over $\mathbb{Z}$, we have $\Phi_m(\lambda) \neq 0$. Therefore, the bitlength of $\Phi_m(\lambda)$ is at least $\log r$, and hence the bitlength of λ is at least $\log r/\varphi(m) \geq \log r/\varphi(k)$. This implies that the shortest Miller loop we can hope to obtain for the ate pairing will have bitlength at least $\log r/\varphi(k)$.

In this chapter, we show how to obtain a pairing which can be computed using Miller functions with bitlength at most $\log r/\varphi(k)$.

3.1. Extended Miller functions

For points P, Q on a curve E, define the function $\ell_{P,Q}$ to be the normalized function with divisor $D_P + D_Q - D_{P+Q}$. Let s be an integer, and let h be a polynomial with integer coefficients, h_i. Define the *extended Miller function*, $f_{s,h,Q}$, as the normalized rational function with divisor

$$\sum_{i=0}^{\deg h} h_i[(s^i Q) - (\infty)].$$

The extended Miller function is a compact way of describing a certain product of Miller functions and lines, based on h. In particular, we have the following relation:

$$f_{s,h,Q} = \prod_{i=0}^{\deg h} f_{h_i, s^i Q} \cdot \prod_{i=1}^{\deg h} \ell_{t_{i-1}Q, h_i s^i Q}$$

where

$$t_i = \sum_{j=0}^{i} h_j s^j.$$

In order to compute $f_{s,h,Q}$, we need to compute $\deg h + 1$ Miller functions of bitlength $\log |h_i|$.

3.1.1. Parameterized curves. For families of curves parameterized by z, we can select a polynomial h so that h_i is written in terms of the curve parameter. That is, we obtain $h_i = h_i(z)$ as polynomials in z. For these curves, we can break up the computation of $f_{s,h,Q}$ into lines and Miller functions with bitlength $\log |z|$. The following lemmas give us an alternative method to compute $f_{s,h,Q}$ for certain values of s, which helps to isolate the Miller functions from s.

Lemma 3.2. *Let $s = ap$, $P \in G_1$, and $Q \in G_2$ for some integer a. Then we can compute $f_{s,h,Q}(P)$ as*

$$f_{s,h,Q}(P) = \prod_{i=0}^{\deg h} f_{h_i, a^i Q}(P)^{p^i} \cdot \prod_{i=1}^{\deg h} \ell_{t_{i-1}Q, h_i a^i \pi^i Q}(P)$$

where

$$t_i = \sum_{j=0}^{i} h_j a^j \pi^j.$$

Proof. By definition, we have

$$\begin{aligned}
f_{s,h,Q}(P) &= \prod_{i=0}^{\deg h} f_{h_i,s^iQ}(P) \cdot \prod_{i=1}^{\deg h} \ell_{t_{i-1}Q,h_is^iQ}(P) \\
&= \prod_{i=0}^{\deg h} f_{h_i,a^ip^iQ}(P) \cdot \prod_{i=1}^{\deg h} \ell_{t_{i-1}Q,h_ia^ip^iQ}(P) \\
&= \prod_{i=0}^{\deg h} f_{h_i,a^i\pi^iQ}(\pi^iP) \cdot \prod_{i=1}^{\deg h} \ell_{t_{i-1}Q,h_ia^i\pi^iQ}(P) \\
&= \prod_{i=0}^{\deg h} f_{h_i,a^iQ}(P)^{p^i} \cdot \prod_{i=1}^{\deg h} \ell_{t_{i-1}Q,h_ia^i\pi^iQ}(P).
\end{aligned}$$

□

Lemma 3.3. Fix $P \in G_1$, $Q \in G_2$. Let $s = aq$, where $q = p^{\mathrm{lcm}(k,d)/d}$ and let ϕ be the degree d automorphism such that $\phi P = qP$ and $\phi Q = q^{-1}Q$. Then we can compute $f_{s,h,P}(Q)$ as

$$f_{s,h,P}(Q) = \prod_{i=0}^{\deg h} f_{h_i,a^iP}(Q)^{q^i} \cdot \prod_{i=1}^{\deg h} \ell_{t_{i-1}P,h_ia^i\phi^iP}(Q)$$

where

$$t_i = \sum_{j=0}^{i} h_ja^j\phi^j.$$

Proof. By definition, we have

$$f_{s,h,P} = \prod_{i=0}^{\deg h} f_{h_i,s^iP} \cdot \prod_{i=1}^{\deg h} \ell_{t_{i-1}P,h_is^iP}.$$

First, we observe that

$$f_{h_i,s^iP} = f_{h_i,a^iq^iP}.$$

Next, we rewrite $f_{h_iq^i,a^iP}$ in two ways

$$\begin{aligned}
f_{h_iq^i,a^iP} &= f_{h_i,a^iq^iP} \cdot f_{q^i,a^iP}^{h_i}, \\
f_{h_iq^i,a^iP} &= f_{h_i,a^iP}^{q^i} \cdot f_{q^i,a^ih_iP}.
\end{aligned}$$

Since $(P, Q) \mapsto f_{q^i,a^iP}(Q)$ is bilinear, we have

$$\begin{aligned}
f_{h_iq^i,a^iP} &= f_{h_i,a^iq^iP} \cdot f_{q^i,a^iP}^{h_i} \\
&= f_{h_i,a^iq^iP} \cdot f_{q^i,a^ih_iP},
\end{aligned}$$

and hence

$$f_{h_i,s^iP} = f_{h_i,a^iq^iP} = f_{h_i,a^iP}^{q^i},$$

from which the result follows. $\square$

Next, we discuss how to exploit the parameterization of h_i to compute $f_{s,h,Q}(P)$ or $f_{s,h,P}(Q)$ using $\max\{\deg h_i(z)\} \cdot \log|z|$ Miller operations and a few extra lines, the number of additional lines depending on the coefficients of the $h_i(z)$'s.

Lemma 3.4. The Miller functions $f_{z,Q}(P)$, $f_{z^2,Q}(P)$, $\ldots$, $f_{z^m,Q}(P)$ can be iteratively computed using $m \log|z|$ Miller operations.

Proof. In the Miller algorithm (Algorithm 2), we begin with the Miller accumulation variable, f, initialized to 1 and end with it containing $f_{z,Q}(P)$. As a side effect, we obtain zQ from the algorithm. If we initialize the next call to the Miller algorithm with $f_{z,Q}(P)$ in the Miller accumulation variable evaluating on the points zQ and P, we obtain $f^z_{z,Q}(P) \cdot f_{z,zQ}(P) = f_{z^2,Q}(P)$. Iteratively, we can obtain $f_{z^3,Q}(P)$, $\ldots$, $f_{z^m,Q}(P)$. $\square$

Lemma 3.4 holds symmetrically for computing $f_{z,P}(Q)$, $f_{z^2,P}(Q)$, $\ldots$, $f_{z^m,P}(Q)$.

Next, we observe that we can use Lemma 3.4 to compute

$$f^{p^i}_{h_i(z),a^iQ}(P)$$

with minimal additional computation.

Lemma 3.5. Set $m = \max\{\deg h_i(z)\}$. Given $f_{z,a^iQ}(P)$, $f_{z^2,a^iQ}(P)$, $\ldots$, $f_{z^m,a^iQ}(P)$, we can compute $f^{p^i}_{h_i(z),a^iQ}(P)$ with little addition computation (depending on the coefficients of $h_i(z)$).

The result holds similarly for $h^{q^i}_{h_i(z),a^iP}(Q)$.

Lemma 3.6. Set $m = max\{\deg h_i(z)\}$. Given $f_{z,a^iP}(Q)$, $f_{z^2,a^iP}(Q)$, $\ldots$, $f_{z^m,a^iP}(Q)$, we can compute $f^{q^i}_{h_i(z),a^iP}(Q)$ with little addition computation (depending on the coefficients of $h_i(z)$).

3.1.2. Optimal pairing results.

The following results show that particular choices of h give extended Miller functions which can be used for pairings. In Section 3.3, we show how to obtain polynomials h which are appropriate for efficient computation. The pairings produced using this method have Miller functions with length approximately $r^{1/\varphi(k)}$. The pairings are "optimal" insofar as they achieve the $r^{1/\varphi(k)}$ bound.

Theorem 3.7 (Vercauteren [87])**.** Let h be a polynomial with integer coefficients h_i such that $h(p) \equiv 0 \pmod r$. Then the following is a pairing

$$(P, Q) \mapsto f_{p,h,Q}(P)^{(p^k-1)/r}.$$

PROOF. Since $h(p) \equiv 0 \pmod r$, we have $mr = h(p)$ for some integer m. Therefore,

$$\begin{aligned} f_{r,Q}^m = f_{mr,Q} = f_{\sum h_i p^i,Q} &= \prod_{i=0}^{\deg h} f_{h_i p^i,Q} \cdot \prod_{i=1}^{\deg h} \ell_{s_{i-1}Q,h_i p^i Q} \\ &= \prod_{i=0}^{\deg h} \left(f_{h_i,p^i Q} \cdot f_{p^i,Q}^{h_i} \right) \cdot \prod_{i=1}^{\deg h} \ell_{s_{i-1}Q,h_i p^i Q} \\ &= f_{p,h,Q} \cdot \prod_{i=0}^{\deg h} f_{p^i,Q}^{h_i} \end{aligned}$$

where

$$s_i = \sum_{j=0}^{i} h_j p^j.$$

Rearranging these terms, we obtain

$$f_{p,h,Q} = f_{r,Q}^m \cdot \prod_{i=0}^{\deg h} f_{p^i,Q}^{-h_i}.$$

Adding in the final exponentiation and evaluating at the point P, we obtain

$$f_{p,h,Q}(P)^{(p^k-1)/r} = \left(f_{r,Q}(P)^{(p^k-1)/r} \right)^m \cdot \prod_{i=0}^{\deg h} \left(f_{p^i,Q}(P)^{(p^k-1)/r} \right)^{-h_i}.$$

By Theorems 2.29 and 2.30, the right-hand side is a product of powers of pairings. Hence, $f_{p,h,Q}(P)^{(p^k-1)/r}$ is a pairing. □

Hess [42] gave a generalization of Vercauteren's Theorem (Theorem 3.7), applying a similar result to the twisted-ate and Weil pairings. We present the theorem in less generality, assuming that the order of the automorphism group, d, divides k, as is the case for many curves, if not all curves, which have been considered for implementation.

Theorem 3.8 (Hess [42, Theorem 1]). Let h be a polynomial with integer coefficients h_i such that $h(q) \equiv 0 \pmod r$ where $q = p^{k/d}$. Then the following are pairings:

$$(P, Q) \mapsto f_{q,h,P}(Q)^{(p^k-1)/r},$$

$$(P, Q) \mapsto \frac{f_{q,h,P}(Q)}{f_{q,h,Q}(P)}.$$

PROOF. Let m be the integer such that $mr = h(q)$. Then

$$\begin{aligned} f_{r,P}^m = f_{mr,P} = f_{\sum h_i q^i, P} &= \prod_{i=0}^{\deg h} f_{h_i q^i, P} \cdot \prod_{i=1}^{\deg h} \ell_{s_{i-1}P, h_i q^i P} \\ &= \prod_{i=0}^{\deg h} \left(f_{h_i, q^i P} \cdot f_{q^i,P}^{h_i} \right) \cdot \prod_{i=1}^{\deg h} \ell_{s_{i-1}P, h_i q^i P} \\ &= f_{q,h,P} \cdot \prod_{i=0}^{\deg h} f_{q^i,P}^{h_i}, \end{aligned}$$

where

$$s_i = \sum_{j=0}^{i} h_j q^j.$$

By rearranging the terms, we obtain that $f_{q,h,P}$ is the product of powers of $f_{r,P}$ and $f_{q^i,P}$. By Theorems 2.29 and 2.30, we have that $f_{q,h,P}(Q)^{(p^k-1)/r}$ is the product of powers of pairings and hence, is a pairing, proving the first part of the theorem.

For the second part, consider that we have obtained

$$f_{q,h,P} = f_{r,P}^m \cdot \prod_{i=0}^{\deg h} f_{q^i,P}^{-h_i}.$$

Symmetrically, from the proof of Theorem 3.7, we can see that

$$f_{q,h,Q} = f_{r,Q}^m \cdot \prod_{i=0}^{\deg h} f_{q^i,Q}^{-h_i}.$$

Combining these observations, we obtain

$$\frac{f_{q,h,P}(Q)}{f_{q,h,Q}(P)} = \left(\frac{f_{r,P}(Q)}{f_{r,Q}(P)} \right)^m \cdot \prod_{i=0}^{\deg h} \left(\frac{f_{q^i,P}(Q)}{f_{q^i,Q}(P)} \right)^{-h_i}.$$

By Theorem 2.27, we have

$$(P, Q) \mapsto \frac{f_{q^i,P}(Q)}{f_{q^i,Q}(P)}$$

is a pairing. Thus, the following is the product of powers of pairings and hence, is itself a pairing

$$(P, Q) \mapsto \frac{f_{q,h,P}(Q)}{f_{q,h,Q}(P)}.$$

□

Although we attribute this theorem to Hess, a less careful reader might claim that Hess's theorem only applies in the case where $k = d$. However, if we take $q = p^{k/d}$ in Hess's theorem, we can see that if the curve E over $\mathbb{F}_p$ has embedding degree k (with respect to r), then the curve E over $\mathbb{F}_q$ has embedding degree d (with respect to r). Hence, the theorem can be applied and the field restricted from $\mathbb{F}_q$ to $\mathbb{F}_p$ to obtain the result of Theorem 3.10.

3.2. Optimal pairings with automorphisms

Let ϕ be a degree-d automorphism of E. Let λ be an integer such that $\phi Q = \lambda Q$ for all $Q \in G_2$. The next lemma gives a relation between an extended Miller function involving λ and other Miller functions.

Lemma 3.9. Let h be a polynomial with integer coefficients h_i such that $h(p\lambda) \equiv 0 \pmod{r}$. Then

$$f_{p\lambda,h,Q} = f_{r,Q}^{m} \cdot \prod_{i=0}^{\deg h} f_{\lambda^i,Q}^{-h_i p^i} \cdot \prod_{i=0}^{\deg h} f_{p^i,\phi^i Q}^{-h_i}.$$

Proof. Let m be the integer such that $mr = h(p\lambda)$. Then

$$\begin{aligned}
f_{r,Q}^{m} &= f_{mr,Q} = f_{\sum h_i (p\lambda)^i,Q} \\
&= \prod_{i=0}^{\deg h} f_{h_i (p\lambda)^i,Q} \cdot \prod_{i=1}^{\deg h} \ell_{s_{i-1}Q,h_i(p\lambda)^i Q} \\
&= \prod_{i=0}^{\deg h} \left(f_{h_i,(p\lambda)^i Q} \cdot f_{(p\lambda)^i,Q}^{h_i} \right) \cdot \prod_{i=1}^{\deg h} \ell_{s_{i-1}Q,h_i(p\lambda)^i Q} \\
&= f_{p\lambda,h,Q} \cdot \prod_{i=0}^{\deg h} f_{(p\lambda)^i,Q}^{h_i} \\
&= f_{p\lambda,h,Q} \cdot \prod_{i=0}^{\deg h} \left(f_{\lambda^i,Q}^{p^i} \cdot f_{p^i,\lambda^i Q} \right)^{h_i} \\
&= f_{p\lambda,h,Q} \cdot \prod_{i=0}^{\deg h} \left(f_{\lambda^i,Q}^{p^i} \cdot f_{p^i,\phi^i Q} \right)^{h_i} \\
&= f_{p\lambda,h,Q} \cdot \prod_{i=0}^{\deg h} f_{\lambda^i,Q}^{h_i p^i} \cdot \prod_{i=0}^{\deg h} f_{p^i,\phi^i Q}^{h_i},
\end{aligned}$$

and the result follows. □

The next lemma allows us to compute a pairing utilizing automorphisms.

Lemma 3.10 (Hess [42, Theorem 2])**.** Let h be a polynomial such that $h(p\lambda) \equiv 0 \pmod{r}$. Then the following is a pairing:

$$(P, Q) \mapsto \left(\prod_{j=0}^{d-1} f_{p\lambda,h,\phi^j Q}(P)^{\lambda^{-j}} \right)^{(p^k-1)/r}.$$

PROOF. By Lemma 3.9, we have

$$f_{p\lambda,h,Q} = f_{r,Q}^{m} \cdot \prod_{i=0}^{\deg h} f_{\lambda^i,Q}^{-h_i p^i} \cdot \prod_{i=0}^{\deg h} f_{p^i,\phi^i Q}^{-h_i}.$$

Evaluated at P and with the inclusion of the final exponentiation, by Lemma 2.10 the terms $f_{p^i,\phi^i Q}^{-h_i}$ give us pairings. By [42, Theorem 2]

$$(P, Q) \mapsto \left(\left(\prod_{j=0}^{d-1} f_{\lambda^i,\phi^j Q}^{\lambda^{-j}}(P) \right)^{-h_i p^i} \right)^{(p^k-1)/r}$$

is a pairing. Hence, we consider

$$\begin{aligned}
\prod_{j=0}^{d-1} f_{p\lambda,h,\phi^j Q}^{\lambda^{-j}} &= \prod_{j=0}^{d-1} \left(f_{r,\phi^j Q}^{m} \cdot \prod_{i=0}^{\deg h} f_{\lambda^i,\phi^j Q}^{-h_i p^i} \cdot \prod_{i=0}^{\deg h} f_{p^i,\phi^j \phi^i Q}^{-h_i} \right)^{\lambda^{-j}} \\
&= \prod_{j=0}^{d-1} f_{r,\phi^j Q}^{m\lambda^{-j}} \cdot \prod_{j=0}^{d-1} \prod_{i=0}^{\deg h} f_{\lambda^i,\phi^j Q}^{-h_i p^i \lambda^{-j}} \cdot \prod_{j=0}^{d-1} \prod_{i=0}^{\deg h} f_{p^i,\phi^j \phi^i Q}^{-h_i \lambda^{-j}} \\
&= \prod_{j=0}^{d-1} f_{r,\phi^j Q}^{m\lambda^{-j}} \cdot \prod_{i=0}^{\deg h} \left(\prod_{j=0}^{d-1} f_{\lambda^i,\phi^j Q}^{\lambda^{-j}} \right)^{-h_i p^i} \cdot \prod_{i=0}^{\deg h} \prod_{j=0}^{d-1} f_{p^i,\phi^j \phi^i Q}^{-h_i \lambda^{-j}}.
\end{aligned}$$

From this, it follows that the map

$$(P, Q) \mapsto \left(\prod_{j=0}^{d-1} f_{p\lambda,h,\phi^j Q}^{\lambda^{-j}}(P) \right)^{(p^k-1)/r}$$

is a product of Tate, automorphism, and ate pairings respectively. □

The key step in the proof of the previous lemma is the translation of $f_{\lambda^i,Q}$ into a pairing by taking the product of pairings evaluated at different $\phi^j Q$. Next, we show how to compute a pairing requiring two (as opposed to d) extended Miller functions.

Theorem 3.11. Let h be a polynomial such that $h(p\lambda) \equiv 0 \pmod{r}$. Let $h_0, \ldots, h_m$ be the coefficients of h. Then the following are pairings:

$$\begin{aligned}
(P, Q) &\mapsto \left(f_{p\lambda,h,Q}(P)^{\lambda+1} \cdot f_{p\lambda,h,\phi Q}(P) \cdot \ell_{\phi Q,Q}^{-h(p)}(P)\right)^{(p^k-1)/r}, && \text{if 3 divides } d,\\
(P, Q) &\mapsto \left(f_{p\lambda,h,Q}(P)^{\lambda} \cdot f_{p\lambda,h,\phi Q}(P)\right)^{(p^k-1)/r}, && \text{if 4 divides } d.
\end{aligned}$$

Proof. By Lemma 3.9, we have

$$f_{p\lambda,h,Q} = f_{r,Q}^{m} \cdot \prod_{i=0}^{\deg h} f_{\lambda^i,Q}^{-h_i p^i} \cdot \prod_{i=0}^{\deg h} f_{p^i,\phi^i Q}^{-h_i}.$$

Evaluating this at P and with the inclusion of the final exponentiation, the terms of the form $f_{p^i,\phi^i Q}^{-h_i}$ give us pairings. We must invoke Theorems 2.31 and 2.32 to turn $f_{\lambda^i,Q}^{-h_i p^i}$ into a pairing. In the case where 3 divides d, we apply Theorem 2.31 to obtain

$$(P, Q) \mapsto \left(\left(f_{\lambda^i,Q}(P)^{\lambda+1} \cdot f_{\lambda^i,\phi Q}(P) \cdot \ell_{\phi Q,Q}(P)\right)^{-h_i p^i}\right)^{(p^k-1)/r}$$

is a pairing. Hence, we consider

$$\begin{aligned}
f_{p\lambda,h,Q}^{\lambda+1} \cdot f_{p\lambda,h,\phi Q} \cdot \ell_{\phi Q,Q}^{-h(p)} &= \left(f_{r,Q}^{m} \cdot \prod_{i=0}^{\deg h} f_{\lambda^i,Q}^{-h_i p^i} \cdot \prod_{i=0}^{\deg h} f_{p^i,\phi^i Q}^{-h_i}\right)^{\lambda+1}\\
&\quad \cdot \left(f_{r,\phi Q}^{m} \cdot \prod_{i=0}^{\deg h} f_{\lambda^i,\phi Q}^{-h_i p^i} \cdot \prod_{i=0}^{\deg h} f_{p^i,\phi^{i+1} Q}^{-h_i}\right) \prod_{i=0}^{\deg h} \ell_{\phi Q,Q}^{-h_i p^i}\\
&= \left(f_{r,Q}^{\lambda+1} \cdot f_{r,\phi Q}\right)^{m} \cdot \prod_{i=0}^{\deg h} \left(f_{p^i,Q}^{\lambda+1} \cdot f_{p^i,\phi Q}\right)^{-h_i}\\
&\quad \cdot \prod_{i=0}^{\deg h} \left(f_{\lambda^i,Q}^{\lambda+1} \cdot f_{\lambda^i,\phi Q} \cdot \ell_{\phi Q,Q}\right)^{-h_i p^i}.
\end{aligned}$$

From this, it follows that the map

$$(P, Q) \mapsto \left(f_{p\lambda,h,Q}^{\lambda+1}(P) \cdot f_{p\lambda,h,\phi Q}(P) \cdot \ell_{\phi Q,Q}^{-h(p)}(P)\right)^{(p^k-1)/r}$$

is a product of Tate, ate, and automorphism pairings respectively.

The proof for 4 dividing d is similar. □

We obtain an analogous result for the Weil paring.

Theorem 3.12. Let h be a polynomial such that $h(\lambda) \equiv 0 \pmod{r}$ where $\lambda Q = \phi Q$ for all $Q \in G_2$ and ϕ is a degree-d automorphism. Then the following is a pairing

$$(P, Q) \mapsto \frac{f_{\lambda,h,P}(Q)}{f_{\lambda,h,Q}(P)}.$$

Proof. The proof is similar to that of the previous theorem. $\square$

3.2.1. Computing the action of automorphisms. All elliptic curves E defined over $\mathbb{F}_p$ satisfy the equation

$$4p - t^2 = Df^2$$

where D is a square-free integer and t is the trace of the p-th power Frobenius. If $D = 1$, then there exists an automorphism of order 4. If $D = 3$, there exists an automorphism of order 3. Using t and f, we show how to compute a square root of $-D$ modulo r.

Lemma 3.13. Let f be invertible modulo r and such that $4p - t^2 = Df^2$. Then

$$(t-2)/f$$

is a square root of $-D$ modulo r.

Proof. We have $p+1 = hr+t$ for some integer h and so $p+1 \equiv t \pmod{r}$. Therefore

$$Df^2 = 4p - t^2 \equiv 4(t-1) - t^2 = -(t-2)^2 \qquad \pmod{r}$$

and so

$$-D = \left(\frac{t-2}{f}\right)^2 \qquad \pmod{r}.$$

Thus, $(t-2)/f$ is a square root of $-D$ modulo r. $\square$

We can use this representation of a square root of $-D$ in order to compute the action of automorphisms on G_2.

Lemma 3.14. Let f be invertible modulo r and such that $4p - t^2 = Df^2$. If $D = 1$, then there exists an order-4 automorphism ϕ of E such that

$$\phi Q = \frac{t-2}{f} Q$$

for all $Q \in G_2$

Proof. Since $(t-2)^2/f^2 \equiv -D = -1 \pmod{r}$, we have that $(t-2)/f$ is distinct from 1 and -1. Since

$$\left(\frac{t-2}{f}\right)^4 \equiv (-D)^2 = D^2 = 1 \qquad \pmod{r},$$

we have that $(t-2)/f$ has order 4 modulo r. Since G_2 is cyclic and the automorphism group of E has order 4, there exists an automorphism ϕ of over 4 which satisfies the lemma. $\square$

Lemma 3.15. Let f be invertible modulo r and such that $4p - t^2 = Df^2$. If $D = 3$ and $r > 2$ is prime, then there exists an order-3 automorphism ϕ such that

$$\phi Q = \left(-\frac{1}{2} - \frac{t-2}{2f}\right) Q$$

for all $Q \in G_2$

Proof. Working modulo r, we have that

$$\left(-\frac{1}{2} - \frac{t-2}{2f}\right)^3 = \left(\frac{-1-\sqrt{-D}}{2}\right)^3 = \left(\frac{-1-\sqrt{-3}}{2}\right)^3 = 1.$$

Since $\left(\frac{t-2}{2f} + \frac{1}{2}\right)$ has order 3 modulo r, G_2 is cyclic, and the automorphism group of E has order 3, there exists an automorphism ϕ such that the lemma is satisfied. $\square$

3.3. Finding nice polynomials h via Minkowski's theorem and the LLL algorithm

In our optimal pairing lemmas, we assumed the existence of polynomials h with certainly properties. For the moment, we focus on the lemmas which require h such that $h(p) \equiv 0 \pmod{r}$. Indeed, many such polynomials are immediately available to us. For example, we can take $h(z) = r$, $h(z) = z - p$, $h(z) = z^2 - p^2$, etc. These choices are not useful, however, since the computational cost of $f_{p,h,Q}$ depends on the magnitude of the coefficients of h. Since the property $h(p) \equiv 0 \pmod{r}$ is preserved under taking integer linear combinations, we consider linear combinations of $h(z) = r$, $h(z) = z - p$, $h(z) = z^2 - p^2$, etc.

The polynomial $h(z) = \Phi_k(z)$ satisfies the conditions and has small coefficients, but produces only degenerate pairings. Therefore, we restrict ourselves to integer linear combinations of $h(z) = r$ and $h(z) = z^i - p^i$ where the degree of h is less than the degree of $\Phi_k(z)$. Such linear combinations correspond to vectors in the lattice spanned by the rows of the matrix

$$M = \left[\begin{array}{l|ccc} r & 0 & \dots & 0 \\ \hline -p & 1 & \dots & 0 \\ \vdots & & \ddots & \\ -p^{\varphi(k)-1} & 0 & \dots & 1 \end{array}\right].$$

The correspondence between vectors in the row space of M and polynomials h is the natural one. Specifically, the vector $\langle h_0, \dots, h_m \rangle$ in the row space of M corresponds to the polynomial with coefficients h_i,

$$h(z) = \sum_{i=0}^{m} h_i z^i,$$

and preserves the property that $h(p) \equiv 0 \pmod{r}$, since the polynomials corresponding to the rows of M satisfy this property.

The following theorem gives an upper bound on the smallest magnitude coefficients of a vector in the row space of M.

Theorem 3.16 (Minkowski's Theorem [62]). Let M be an n-row integer matrix with volume $v = |\det M|$. There exists a vector in the row space of M with coordinates having magnitude at most $v^{1/n}$.

By Theorem 3.16, there exists a polynomial h of degree less than $\varphi(k)$ with coefficients whose magnitudes have bitlength less than $\log r/\varphi(k)$. When we explore examples with parameterized curves, we can usually obtain a nice solution manually performing row operations using Maple. In the case of non-parameterized curves, by using the following algorithm by Lenstra *et al.* we can obtain an explicit h.

Theorem 3.17 (The LLL algorithm [53]). Let M be an n-row integer matrix with volume $v = |\det M|$. We can efficiently compute a vector in the rowspace of M such that the coordinates have magnitude at most $2^{(n-1)/4} \cdot v^{1/n}$.

The LLL algorithm actually produces a basis of the row space such that the product of the norms of the basis elements is less than $2^{n(n-1)/4} \cdot v$. It follows that there exists a basis element with norm at most $2^{(n-1)/4} \cdot v^{1/n}$ and hence the coordinates have magnitude at most $2^{(n-1)/4} \cdot v^{1/n}$.

3.4. Examples

In this section, we use Minkowski's Theorem to illustrate the optimal pairing results for parameterized elliptic curves.

3.4.1. BN curves. BN curves [10] have embedding degree $k = 12$ and are parameterized by z such that

$$r = r(z) = 36z^4 + 36z^3 + 18z^2 + 6z + 1$$
$$p = p(z) = 36z^4 + 36z^3 + 24z^2 + 6z + 1$$

are both prime.

To construct an optimal Tate pairing using Theorem 3.7, we need to determine a polynomial h such that $h(p) \equiv 0 \pmod{r}$. Elements in the rowspace of the following matrix give us such polynomials

$$M = \begin{bmatrix} r(z) & 0 & 0 & 0 \\ -p(z) & 1 & 0 & 0 \\ -p(z)^2 & 0 & 1 & 0 \\ -p(z)^3 & 0 & 0 & 1 \end{bmatrix}.$$

The coefficients of the polynomial h will be polynomials in z. By Minkowski's Theorem, we hope to find h with coefficients having degree at most 1 in xz, since $\deg r/\varphi(k) = 4/4 = 1$. As a first step, we substitute the polynomials p and r and reduce the last three entries in the first column modulo r. This corresponds to integer row operations and yields

$$\begin{bmatrix} 36z^4 + 36z^3 + 18z^2 + 6z + 1 & 0 & 0 & 0 \\ -6z^2 & 1 & 0 & 0 \\ 36z^3 + 18z^2 + 6z + 1 & 0 & 1 & 0 \\ 36z^3 + 24z^2 + 12z + 3 & 0 & 0 & 1 \end{bmatrix}.$$

Next, we add $\mathbb{Z}[z]$-multiples of rows with smaller degrees in the first column to those with larger degrees. For example, we can subtract row 3 from row 4 and subtract the x-multiple of row 3 from row 1 to obtain

$$\begin{bmatrix} 18z^3 + 12z^2 + 5z + 1 & 0 & 0 & 0 \\ -6z^2 & 1 & 0 & 0 \\ 36z^3 + 18z^2 + 6z + 1 & 0 & 1 & 0 \\ 6z^2 + 6z + 2 & 0 & -1 & 1 \end{bmatrix}.$$

Continuing this process, we obtain a matrix with only linear entries

$$\hat{M} = \begin{bmatrix} -z & 2z & 2z+1 & -z \\ 2z & z+1 & -z & z \\ -1 & 2z+1 & 1 & 2z \\ 2z+1 & 0 & 2z & 1 \end{bmatrix}.$$

The rows of the reduced matrix $\hat{M}$ correspond to the following polynomials

$$\begin{array}{lrcrcrcr} x \mapsto & (-z) & + & (2z)x & + & (2z+1)x^2 & + & (-z)x^3 \\ x \mapsto & (2z) & + & (z+1)x & + & (-z)x^2 & + & (z)x^3 \\ x \mapsto & (-1) & + & (2z+1)x & + & (1)x^2 & + & (2z)x^3 \\ x \mapsto & (2z+1) & + & (0)x & + & (2z)x^2 & + & (1)x^3. \end{array}$$

The fourth polynomial seems like a good choice, since is has a coefficient of 0 on the x term. Let $h(x)$ be this polynomial. Using Lemma 3.2, we can write the corresponding extended Miller function as follows:

$$f_{p,h,Q} = f_{2z+1,Q} \cdot f_{2z,Q}^{p^2} \cdot f_{1,Q}^{p^3} \cdot \ell_{(2z+1)Q,2zp^2Q} \cdot \ell_{(2z+1+2zp^2)Q,p^3Q}.$$

At first, it appears as though we need to compute two Miller loops of lengths $2z+1$ and $2z$. However, we can use Lemma 3.5 to reduce the equation to one Miller function and a few extra lines. Under the equivalence induced by the final exponentiation we obtain

$$\ell_{(2z+1+2zp^2)Q,p^3Q} = \ell_{(h(p)-p^3)Q,p^3Q} = \ell_{-p^3Q,p^3Q} = v_{p^3Q} \equiv 1.$$

Also, we observe that $f_{1,p^3Q} = 1$. From these observations, we can compute the pairing as

$$\begin{aligned} f_{p,h,Q} &= f_{2z+1,Q} \cdot f_{2z,Q}^{p^2} \cdot f_{1,Q}^{p^3} \cdot \ell_{(2z+1)Q,2zp^2Q} \cdot \ell_{(2z+1+2zp^2)Q,p^3Q} \\ &= f_{2z,Q} \cdot \ell_{2zQ,Q} \cdot f_{2z,Q}^{p^2} \cdot \ell_{(2z+1)Q,2zp^2Q} \\ &\equiv f_{2z,Q}^{p^2+1} \cdot \ell_{2zQ,Q} \cdot \ell_{(2z+1)Q,2z\pi^2Q}. \end{aligned}$$

This completes a construction of an optimal Tate pairing for BN curves.

Next, we construct an optimal Weil pairing using Theorem 3.12. We expect to find a polynomial h with coefficients having degree $\deg r/2 = 2$ such that $h(p^2) \equiv 0 \pmod{r}$. After finding such a polynomial, we will construct not only an optimal Weil pairing but an optimal twisted pairing as well.

We begin by reducing the follow matrix

$$\begin{aligned} M &= \begin{bmatrix} r(z) & 0 \\ -p(z)^2 & 1 \end{bmatrix} \\ &\equiv \begin{bmatrix} 36z^4+36z^3+18z^2+6z+1 & 0 \\ 36z^3+18z^2+6z+1 & 1 \end{bmatrix} \\ &\equiv \begin{bmatrix} 18z^3+12z^2+5z+1 & -z \\ 36z^3+18z^2+6z+1 & 1 \end{bmatrix} \\ &\equiv \begin{bmatrix} 18z^3+12z^2+5z+1 & -z \\ -6z^2-4z-1 & 2z+1 \end{bmatrix} \\ &\equiv \begin{bmatrix} 2z+1 & 6z^2+2z \\ -6z^2-4z-1 & 2z+1 \end{bmatrix}. \end{aligned}$$

All entries in the reduced matrix M have degree-in-z at most 2 but the first row has fewer terms. Set $h(x) = (2z+1)+(6z^2+2z)x$ and observe that we can ignore the following line:

$$\ell_{(2z+1)Q,(6z^2+2z)p^2Q} = \ell_{(2z+1)Q,-(2z+1)Q} = v_{(2z+1)Q}.$$

From Lemma 3.2 and Lemma 3.3, we have

$$\begin{aligned} f_{p^2,h,P} &\equiv f_{2z+1,P} \cdot f_{6z^2+2z,P}^{p^2} \\ f_{p^2,h,Q} &\equiv f_{2z+1,Q} \cdot f_{6z^2+2z,Q}^{p^2}. \end{aligned}$$

This gives us the pairing

$$(P,Q) \mapsto \frac{f_{2z+1,P}(Q)\cdot f_{6z^2+2z,P}(Q)^{p^2}}{f_{2z+1,Q}(P)\cdot f_{6z^2+2z,Q}(P)^{p^2}}. \tag{3.1}$$

From Lemma 3.5, we are able to compute $f_{2z,Q}$ as an intermediate step in computing $f_{6z^2,Q}$ allowing us to effectively compute $f_{p^2,h,Q}$ using a Miller function of length $6z^2$ and a few extra lines. Similarly, we can compute $f_{2z,P}$ as an intermediate step of computing $f_{6z^2,P}$ by Lemma 3.6. Therefore, we can compute $f_{2z,P}$, $f_{2z,Q}$, $f_{6z^2,P}$, $f_{6z^2,Q}$ using approximately $2\log z$ Miller operations and using these values, we can perform the following computations

$$\begin{aligned}
f_{p^2,h,P} &\equiv f_{2z+1,P}\cdot f_{6z^2+2z,P}^{p^2}\\
&= f_{2z,P}\cdot \ell_{2zP,P}\cdot f_{6z^2,P}^{p^2}\cdot f_{2z,P}^{p^2}\cdot \ell_{6z^2P,2zP}\\
&= f_{2z,P}^{p^2+1}\cdot f_{6z^2,P}^{p^2}\cdot \ell_{2zP,P}\cdot \ell_{6z^2P,2zP}^{p^2},\\
f_{p^2,h,Q} &\equiv f_{2z,Q}^{p^2+1}\cdot f_{6z^2,Q}^{p^2}\cdot \ell_{2zQ,Q}\cdot \ell_{6z^2Q,2zQ}^{p^2}.
\end{aligned}$$

This gives us the following pairing

$$(P,Q)\mapsto \left(\frac{f_{2z,P}(Q)}{f_{2z,Q}(P)}\right)^{p^2+1}\left(\frac{f_{6z^2,P}(Q)\cdot \ell_{6z^2P,2zP}(Q)}{f_{6z^2,Q}(P)\cdot \ell_{6z^2Q,2zQ}(P)}\right)^{p^2}\frac{\ell_{2zP,P}(Q)}{\ell_{2zQ,Q}(P)}.$$

The numerators of this optimal Weil pairing along with the final exponentiation, give us an optimal twisted pairing

$$(P,Q)\mapsto \left(f_{2z,P}^{p^2+1}(Q)\cdot f_{6z^2,P}^{p^2}(Q)\cdot \ell_{2zP,P}(Q)\cdot \ell_{6z^2P,2zP}^{p^2}(Q)\right)^{(p^k-1)/r}.$$

3.4.2. BLS-12 curves. BLS-12 curves [8] (see also [19]) have embedding degree $k = 12$ and are parameterized by z such that

$$\begin{aligned}
r &= r(z) = z^4 - z^2 + 1,\\
p &= p(z) = \frac{1}{3}(z-1)^2(z^4 - z^2 + 1) + z,
\end{aligned}$$

are both prime.

The curve has order $\#E(\mathbb{F}_p) = (z-1)^2(z^4 - z^2 + 1)/3$ and hence the trace of the p-th power Frobenius is $t(z) = z+1$. From Minkowski's theorem, we expect we could determine an h with coefficients which are linear in z. Since the ate pairing with $z \equiv p \pmod{r}$ has Miller length z, we have already achieved this bound. Thus, we need not bother with the optimal pairing framework to derive an optimal Tate pairing. Similarly, we have $z^2 \equiv p^{k/d}$

(mod r) and thus the optimal Weil and twisted pairings don't need to be considered. We give all three pairings explicitly:

$$\begin{aligned}(P,Q) &\mapsto f_{z,Q}(P)^{(p^k-1)/r}\\ (P,Q) &\mapsto f_{z^2,P}(Q)^{(p^k-1)/r}\\ (P,Q) &\mapsto \frac{f_{z^2,P}(Q)}{f_{z^2,Q}(P)}.\end{aligned}$$

3.4.3. KSS-18 curves. KSS-18 curves [47] have embedding degree $k = 18$ and are parameterized by z such that

$$\begin{aligned}&r = r(z) = (z^6 + 37z^3 + 343)/343,\\ &p = p(z) = (z^8 + 5z^7 + 7z^6 + 37z^5 + 188z^4 + 259z^3 + 343z^2 + 1763z + 2401)/21,\end{aligned}$$

are both prime.

The parameters of this family of curves give us potentially much more complicated values for h. We begin by reducing the matrix M:

$$M = \begin{bmatrix} r(z) & 0 & 0 & 0 & 0 & 0\\ -p(z) & 1 & 0 & 0 & 0 & 0\\ -p(z)^2 & 0 & 1 & 0 & 0 & 0\\ -p(z)^3 & 0 & 0 & 1 & 0 & 0\\ -p(z)^4 & 0 & 0 & 0 & 1 & 0\\ -p(z)^5 & 0 & 0 & 0 & 0 & 1\end{bmatrix} \equiv \begin{bmatrix} (z^6+37z^3+343)/343 & 0 & 0 & 0 & 0 & 0\\ (-z^4-16z)/7 & 1 & 0 & 0 & 0 & 0\\ (5z^5+87z^2)/49 & 0 & 1 & 0 & 0 & 0\\ z^3+18 & 0 & 0 & 1 & 0 & 0\\ (-3z^4-55z)/7 & 0 & 0 & 0 & 1 & 0\\ (8z^5+149z^2)/49 & 0 & 0 & 0 & 0 & 1\end{bmatrix}.$$

We notice that all of the entries in the first column have (integer) denominators. This is a side effect of the curve construction. We have already restricted ourselves to parameters z such that $r(z)$, $p(z)$ are prime and as a result, the first column will always take on integer values for these values of z. Hence, the denominators will not pose a problem as long as we keep to integer linear combinations.

Eventually, we obtain the vector

$$\langle z, 3, 0, 0, -1, 0\rangle$$

which corresponds to the polynomial $h(x) = z + 3x - x^4$. We may compute $f_{p,h,Q}$ as

$$f_{p,h,Q} = f_{z,Q} \cdot f_{3,Q}^p \cdot \ell_{zQ,(3p-p^4)Q} \cdot \ell_{3pQ,-p^4Q}.$$

Since $\ell_{zQ,(3p-p^4)Q} = \ell_{zQ,-zQ} = v_{zQ}$, it will be eliminated by the final exponentiation. Therefore, the following is an optimal Tate pairing

$$(P,Q) \mapsto (f_{z,Q}(P) \cdot f_{3,Q}^p(P) \cdot \ell_{3\pi Q,-\pi^4 Q}(P))^{(p^k-1)/r}.$$

Next, we turn our attention to optimal Weil pairings. Since $p^3 \equiv z^3 + 18 \pmod{r}$, the optimal pairing framework is not needed, since the following Weil pairing meets the expected optimal pairing Miller length

$$(P, Q) \mapsto \frac{f_{z^3+18,P}(Q)}{f_{z^3+18,Q}(P)}.$$

3.4.4. SB-6 curves. SB-6 curves [81] have embedding degree $k = 6$. We wish to restrict ourselves to the subset of the curves possessing an automorphism of order $d = 4$. The subset of curves with order-4 automorphisms is parameterized by z such that

$$r = r(z) = 16z^8 - 32z^6 + 12z^4 + 4z^2 + 1,$$
$$p = p(z) = 4z^{10} - 8z^8 + 3z^6 - 3z^4 + \frac{17}{4}z^2 + 1,$$

are both prime. SB-6 curves are the first example we have encountered where d does not divide k. Hence, we will illustrate the utility of some of the automorphism-related optimal pairing results.

Since $p \equiv -4z^4 + 4z^2 + 1 \pmod{r}$, the following ate pairing meets the optimal pairing bound

$$(P, Q) \mapsto f_{-4z^4+4z^2+1,Q}(P)^{(p^k-1)/r}.$$

Next, we leverage automorphisms in our pairing computation. Since

$$4p(z) - t(z)^2 = (4z^5 - 6z^3 + z)^2,$$

we have, by Lemma 3.14, that

$$\frac{t(z) - 2}{4z^5 - 6z^3 + z} \equiv 8z^7 - 12z^5 + 2z^3 + 2z \pmod{r(z)}$$

is the action of some order-4 automorphism ϕ on G_2.

Setting $\lambda = 8z^7 - 12z^5 + 2z^3 + 2z$, we construct the following lattice and reduce it until we obtain a vector meeting the Minkowski bound:

$$\begin{aligned} M &= \begin{bmatrix} r(z) & 0 & 0 & 0 \\ -p(z)\lambda(z) & 1 & 0 & 0 \\ -p(z)^2\lambda(z)^2 & 0 & 1 & 0 \\ -p(z)^3\lambda(z)^3 & 0 & 0 & 1 \end{bmatrix} \\ &\equiv \begin{bmatrix} 16z^8 - 32z^6 + 12z^4 + 4z^2 + 1 & 0 & 0 & 0 \\ -8z^7 + 16z^5 - 8z^3 - z & 1 & 0 & 0 \\ -4z^4 - 4z^2 & 0 & 1 & 0 \\ -8z^7 + 12z^5 - 2z^3 - 2z & 0 & 0 & 1 \end{bmatrix} \\ &\equiv \begin{bmatrix} -2z^2+1 & 2z & -1 & 0 \\ -2z & 2z^2 & 0 & -2z^2+1 \\ 2z^2 & -2z & -2z^2+1 & 2z \\ 0 & 2z^2-1 & -2z & 1 \end{bmatrix}. \end{aligned}$$

The first row looks good enough. So we take $h(x) = (-2z^2+1) + (2z)x - x^2$ and apply Theorem 3.11 to obtain the following pairing

$$(P, Q) \mapsto \left(f_{p\lambda,h,Q}(P)^{\lambda} \cdot f_{p\lambda,h,\phi Q}(P)\right)^{(p^k-1)/r}.$$

Turning our attention to the Weil pairing construction, reduce the following matrix

$$\begin{aligned} M &= \begin{bmatrix} r(z) & 0 \\ -\lambda(z) & 1 \end{bmatrix} \\ &= \begin{bmatrix} 16z^8 - 32z^6 + 12z^4 + 4z^2 + 1 & 0 \\ -8z^7 + 12z^5 - 2z^3 - 2z & 1 \end{bmatrix} \\ &\equiv \begin{bmatrix} 4z^4 - 6z^2 + 1 & -4z^3 + 4z \\ 4z^3 - 4z & 4z^4 - 6z^2 + 1 \end{bmatrix}. \end{aligned}$$

Setting $h(x) = (4z^4 - 6z^2 + 1) + (-4z^3 + 4z)x$ and applying Theorem 3.12, we obtain the following Weil pairing

$$(P, Q) \mapsto \frac{f_{\lambda,h,P}(Q)}{f_{\lambda,h,Q}(P)}.$$

3.4.5. BLS-8 curves. BLS-8 curves [8] have embedding degree $k = 8$ and an automorphism of order $d = 3$. These curves are parameterized by z such that

$$\begin{aligned} r &= r(z) = z^8 - z^4 + 1, \\ p &= p(z) = \frac{1}{3}(z-1)^2(z^8 - z^4 + 1) + z^9, \end{aligned}$$

are both prime. These curves are such that d does not divide k.

To construct an optimal pairing, we reduce the following lattice

$$M = \begin{bmatrix} r(z) & 0 & 0 & 0 \\ -p(z) & 1 & 0 & 0 \\ -p(z)^2 & 0 & 1 & 0 \\ -p(z)^3 & 0 & 0 & 1 \end{bmatrix} \equiv \begin{bmatrix} z^8 - z^4 + 1 & 0 & 0 & 0 \\ -z^5 + z & 1 & 0 & 0 \\ z^6 & 0 & 1 & 0 \\ -z^3 & 0 & 0 & 1 \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 & -z^2 & -z \\ z & 1 & 0 & -z^2 \\ z^2 & z & 1 & 0 \\ 0 & z^2 & z & 1 \end{bmatrix}.$$

The third row looks promising, so we set $h(x) = (z^2) + (z)x + x^2$ and obtain the following optimal pairing

$$(P, Q) \mapsto f_{p,h,Q}(P)^{(p^k-1)/r}.$$

Now, we show how to use automorphisms to reduce the Miller loop length. We have

$$4p(z) - t(z)^2 = 3\left(\frac{1}{3}(z^2 + z - 1)(z^3 + z^2 + 1)\right)^2.$$

By Lemma 3.15, there exists an order-3 automorphism ϕ such that $\phi Q = \lambda(z)Q$, where

$$\lambda(z) \equiv \frac{1}{2}\left(-1 - \frac{t(z) - 2}{(z^2 + z - 1)(z^3 + z^2 + 1)/3}\right) \equiv z^4 \pmod{r(z)}.$$

Since $p(z)\lambda(z) \equiv z \pmod{r}$, we can take $h(x) = (-z) + x$ and obtain the following pairing via Theorem 3.11:

$$(P, Q) \mapsto \left(f_{p\lambda,h,Q}(P)^{\lambda+1} \cdot f_{p\lambda,h,\phi Q}(P) \cdot \ell_{\phi Q,Q}^{-h(p)}(P)\right)^{(p^k-1)/r}.$$

To obtain an optimal Weil pairing using automorphisms, we set $h(z) = (x^4) - z$ and apply Theorem 3.12 to obtain the following Weil pairing:

$$(P, Q) \mapsto \frac{f_{\lambda,h,P}(Q)}{f_{\lambda,h,Q}(P)}.$$

3.5. Summary of optimal pairing results

Theorem 3.18 (Optimal Tate)**.** There exists a polynomial h of degree $\varphi(k)$ with coefficients h_i such that $|h_i| \leq r^{1/\varphi(k)}$ and the following is a pairing

$$(P, Q) \mapsto f_{p,h,Q}(P)^{(p^k-1)/r}.$$

Theorem 3.19 (Optimal Weil and twisted ate)**.** There exists a polynomial h of degree $\varphi(d) \in \{1, 2\}$ with coefficients h_i such that $|h_i| \leq r^{1/\varphi(d)}$ and the following are pairings

$$(P, Q) \mapsto f_{q,h,P}(Q)^{(p^k-1)/r},$$

$$(P, Q) \mapsto \frac{f_{q,h,P}(Q)}{f_{q,h,Q}(P)}.$$

Theorem 3.20 (Optimal Tate with automorphisms)**.** Let 3 (or 4) divide d and 3 (respectively 4) not divide k. There exists a polynomial h of degree $2\varphi(k)$ with coefficients h_i such that $|h_i| \leq r^{1/2\varphi(k)}$ and the following is a pairing

$$(P, Q) \mapsto \left(\prod_{i=0}^{d-1} f_{p\lambda,h,\phi^i Q}(P)^{\lambda^{-i}} \right)^{(p^k-1)/r}.$$

Theorem 3.21 (Optimal Weil with automorphisms)**.** Let ϕ be a degree-d automorphism of E and λ be such that $\lambda Q = \phi Q$ for all $Q \in G_2$. There exists a polynomial h of degree $\varphi(d) \in \{1, 2\}$ with coefficients h_i such that $|h_i| \leq r^{1/\varphi(d)}$ and the following is a pairing

$$(P, Q) \mapsto \frac{f_{\lambda,h,P}(Q)}{f_{\lambda,h,Q}(P)}.$$

Theorem 3.22 (Improved optimal Tate with automorphisms)**.** Let 3 (or 4) divide d and 3 (respectively 4) not divide k. Let ϕ be a degree-d automorphism of E and λ be such that $\lambda Q = \phi Q$ for all $Q \in G_2$. Then there exists a polynomial h of degree $2\varphi(k)$ with coefficients h_i such that $|h_i| \leq r^{1/2\varphi(k)}$ and one of the following is a pairing

$$(P, Q) \mapsto \left(f_{p\lambda,h,Q}(P)^{\lambda+1} \cdot f_{p\lambda,h,\phi Q}(P) \cdot \ell_{\phi Q,Q}^{-h(p)}(P) \right)^{(p^k-1)/r}, \qquad \text{if 3 divides } d,$$

$$(P, Q) \mapsto \left(f_{p\lambda,h,Q}(P)^{\lambda} \cdot f_{p\lambda,h,\phi Q}(P) \right)^{(p^k-1)/r}, \qquad \text{if 4 divides } d.$$

CHAPTER 4

Parallel Implementation of Pairings

In this chapter, we continue the work initiated by Grabher, Großschädl and Page [38] on implementing pairings on multi-core platforms. This work is especially challenging for asymmetric pairings because of the apparent paucity of opportunities available for parallelizing Miller's basic algorithm. In particular, it seems hopeless to expect the optimal 2-fold speedup for known algorithms when going from 1 core to 2 cores on existing computing platforms. Furthermore, effective usage of parallel computation resources depends on expensive operating system calls for thread creation and synchronization and on the relative immaturity of development tools such as compilers, profilers, and debuggers. Concurrent programming is difficult in general, due to the fundamentally nondeterministic nature of the multi-threading programming model [52], but it becomes even harder when the computational cost of what is being computed is not several orders of magnitude higher than the parallelization overhead itself.

We focus our attention on the fastest-known asymmetric pairing at the 128-bit security level, namely Vercauteren's optimal ate pairing over BN elliptic curves (Section 3.4.1). We exploit a method introduced in [3] for parallelizing a Miller function evaluation and a new 'delayed squaring' technique. Our implementation is about 1.23 times faster using two cores of an Intel Core i5 or Core i7 machine, and 1.45 times faster using 4 cores of the Core i7 than the state-of-the-art implementation on a single core [2]. We observe that the straightforward methods for parallelizing extension field multiplication that were deemed effective in [38] fail on the platforms under consideration because our field multiplication is much faster, whereas the cost of managing the resulting threads dominates the cost of useful computation.

The limited success in parallelizing the optimal ate pairing on BN curves is due in part to the apparent difficulty in parallelizing the final exponentiation. This motivated us to consider the Weil pairing, whose potential speed advantages over the Tate pairing due to the absence of a final exponentiation in the former were first considered in [50]. We study two optimal Weil pairings, both of which can be computed using the equivalent of four independent Miller functions each having optimal length, and without an expensive final exponentiation. The first pairing is the Weil pairing instantiation we gave in Section 3.4.1 of Hess's general Weil pairing construction [42], while the second pairing is an elegant new construction tailored for parallel execution. These pairings are faster than previous variants of the Weil pairing proposed in [92] and [42]. Our experimental results suggest that the new Weil pairing is 1.25 times faster than the optimal ate pairing on 8-core extensions of the Intel Core i5 and Core i7 machines.

We emphasize that our implementations are for a *single* pairing evaluation on multiple cores. If a protocol requires multiple pairing evaluations, the best strategy may be to simply execute each pairing on a single core of a multi-core platform — the optimal strategy depends on several factors including the number of available cores. Thus, our work is primarily directed at protocols that require a single pairing evaluation in applications that have stringent response time requirements or where the processing power of individual cores in a multi-core platform is low. Some examples of protocols that require a single pairing evaluation are the encryption and decryption procedures in the Boneh-Franklin identity-based encryption scheme (Section 1.1.3), signature verification in the Boneh-Boyen short signature scheme (Section 1.3.1), and Scott's identity-based key agreement scheme (Section 1.3.2).

4.1. Parallelizing the optimal ate pairing

This section shows how the Miller function $f_{s,R}$ in the optimal ate pairing can be split into shorter Miller functions, which can then be computed on separate cores. We shall assume that all Miller functions and line functions are normalized. Equations involving Miller and line functions hold up to multiplication by nonzero constants. Recall the following two properties of Miller functions.

Lemma 4.1 (Miller [61])**.** Let a and b be non-negative integers, and let $R \in E(\mathbb{F}_{q^k})$. Then

(i) $f_{a+b,R} = f_{a,R} \cdot f_{b,R} \cdot \ell_{aR,bR}$; and
(ii) $f_{ab,R} = f_{b,R}^{a} \cdot f_{a,bR}$.

The method of [3] for parallelizing the computation of a Miller function $f_{s,R}$ is the following. We first write $s = 2^w s_1 + s_0$, where $s_0 < 2^w$. Applying Lemma 4.1, we obtain

$$f_{s,R} = f_{s_1,R}^{2^w} \cdot f_{2^w,s_1R} \cdot f_{s_0,R} \cdot \frac{\ell_{2^w s_1 R, s_0 R}}{v_{sR}}. \tag{4.1}$$

If s_0 is small, then the Miller function $f_{s_0,R}$ can be computed relatively cheaply. Thus the computation of $f_{s,R}$ can be parallelized by computing $f_{s_1,R}^{2^w}$ on one processor and f_{2^w,s_1R} on a second processor. The parameter w should be carefully selected in order to balance the time of the two function computations. The relevant criteria for selecting w include the Hamming weight of s_1 (which determines the number of additions in the Miller loop for the first function), and the cost of the w-fold squaring in the first function relative to the cost of computing s_1R in the second function. The w-fold squaring in $f_{s_1,R}^{2^w}$ can be sped up by first computing

$$\alpha = f_{s_1,R}^{(p^6-1)(p^2+1)}$$

(recall that exponentiation by $(p^6-1)(p^2+1)$ is the easy part of the final exponentiation), followed by α^{2^w}. The advantage of this 'delayed squaring' trick is that α belongs to the order-$(p^4 - p^2 + 1)$ cyclotomic subgroup of $\mathbb{F}_{p^{12}}^*$ whence Karabina's squaring method [48] (see also [39]) can be deployed at a cost of 12 $\mathbb{F}_p$ multiplications plus some small overhead — this is considerably less than squaring a general element in $\mathbb{F}_{p^{12}}$ which costs 24 $\mathbb{F}_p$ multiplications.

Each of the two expensive Miller function computations in (4.1) can be recursively parallelized. For this purpose, one writes

$$s = s_t 2^{w_t} + \cdots + s_2 2^{w_2} + s_1 2^{w_1} + s_0,$$

where

$$s_i 2^{w_i} = (s \bmod 2^{w_{i+1}}) - (s \bmod 2^{w_i})$$

for some $w_t > \cdots > w_2 > w_1 > w_0 = 0$.

Remark 4.1. We were unable to find any effective method for parallelizing the hard part of the final exponentiation. The exponent $(p^4 - p^2 + 1)/r$ can be decomposed into a multi-addition chain requiring the consecutive z-th powers α^z, α^{z^2} and α^{z^3} where $\alpha \in \mathbb{F}_{p^{12}}$ and z is the BN parameter [84]. However, the extremely low Hamming weight of z limits the potential for parallelization. Furthermore, techniques that exploit very fast squaring (e.g., [89]) and fixed bases (e.g., [55]) are not applicable.

4.2. Optimal Weil pairings

This section presents two Weil pairings, called the α (Section 4.2.1) and β (Section 4.2.2) pairings, that are well suited for parallelization. In this section, E is a BN curve defined over $\mathbb{F}_p$ with BN parameter z. Unless otherwise stated, all functions are assumed to be normalized. By a 'pairing' we will mean a non-degenerate bilinear pairing from $G_1 \times G_2$ to G_T. Note that if e is a pairing and $\gcd(\ell, r) = 1$, then e^ℓ is also a pairing. It is understood that pairing values are defined to be 1 if either input point is equal to ∞.

The classical Weil pairing (Section 2.5.1) is $w_r : G_1 \times G_2 \to G_T$ defined by

$$w_r : (P, Q) \mapsto -\frac{f_{r,P}(Q)}{f_{r,Q}(P)}. \tag{4.2}$$

Note that the Weil pairing does not have a final exponentiation. The two Miller functions in (4.2) each have length approximately z^4 and can be independently computed on two processors.

4.2.1. The α Weil pairing. Recall our construction from Section 3.4.1 (Equation 3.1), which produced the following pairing over $G_1 \times G_2$

$$\alpha \colon (P, Q) \mapsto \left(\frac{f_{2z+1,P}(Q)}{f_{2z+1,Q}(P)} \cdot \frac{f_{6z^2+2z,P}(Q)^{p^2}}{f_{6z^2+2z,Q}(P)^{p^2}} \right)^{(p^6-1)(p^2+1)}. \tag{4.3}$$

The exponentiation by $(p^6-1)(p^2+1)$ provides us with several advantages. In particular, the four Miller functions in (4.3) only need to be semi-normalized and the important denominator elimination speedup can be applied (see Section 2.5.3). Furthermore, the delayed squaring technique of Section 4.1 can be employed as described below. In order to shorten the length of the Miller functions $f^{p^2}_{6z^2+2z,R}$ for $R \in \{P, Q\}$ in (4.3), we can use Lemma 4.1(ii) to write

$$f_{6z^2+2z,R} = f_{z,(6z+2)R} \cdot f^z_{6z+2,R}.$$

Thus, we obtain the following alternate formulation of the α pairing:

$$\alpha : (P,Q) \mapsto \left(\frac{f_{2z+1,P}(Q)}{f_{2z+1,Q}(P)} \cdot \left(\frac{f_{z,(6z+2)P}(Q) \cdot f^z_{6z+2,P}(Q)}{f_{z,(6z+2)Q}(P) \cdot f^z_{6z+2,Q}(P)} \right)^{p^2} \right)^{(p^6-1)(p^2+1)}. \tag{4.4}$$

The revised formula for the α pairing now has six Miller functions, and it may appear that at least six processors would be necessary to effectively parallelize the pairing. However, if $f_{z,R}$ is computed first, then $f_{2z+1,R}$ and $f_{6z+2,R}$ can be computed thereafter with little additional work. Thus, there are effectively only four Miller functions in (4.4). Each of these Miller functions has length approximately z, and therefore the α pairing is considered optimal in the sense of Vercauteren (cf. Theorem 3.7).

Figure 4.1 illustrates the execution path when the four Miller functions in (4.4) are computed in parallel using four processors. A further optimization is to raise the Miller functions to the power $(p^6-1)(p^2+1)$ as soon as they are computed — this enables the use of Karabina's fast squaring when computing $f^z_{6z+2,P}(Q)$ and $f^z_{6z+2,Q}(P)$. Note also that since

$$(6z+2) + p - p^2 + p^3 \equiv 0 \pmod{r},$$

we have

$$(6z+2)Q = -\pi(Q) + \pi^2(Q) - \pi^3(Q)$$

and thus $(6z+2)Q$ can be computed very quickly. The fourth processor in Figure 4.1 is the bottleneck because of the exponentiation by z.

$$\begin{tikzcd}
1. & (6z+2)P \arrow[rr] & & f^{p^2}_{z,(6z+2)P}(Q) \arrow[r] & f_{2z+1,P}(Q) \cdot f^{p^2}_{6z^2+2z,P}(Q) \\
2. & f_{2z+1,P}(Q) \arrow[r] \arrow[rrr, bend right] & f_{6z+2,P}(Q) \arrow[r] & f^z_{6z+2,P}(Q) \arrow[r] & f_{2z+1,P}(Q) \cdot f^{zp^2}_{6z+2,P}(Q) \arrow[u] \\
3. & (6z+2)Q \arrow[rr] & & f^{p^2}_{z,(6z+2)Q}(P) \arrow[r] & f_{2z+1,Q}(P) \cdot f^{p^2}_{6z^2+2z,Q}(P) \\
4. & f_{2z+1,Q}(P) \arrow[r] \arrow[rrr, bend right] & f_{6z+2,Q}(P) \arrow[r] & f^z_{6z+2,Q}(P) \arrow[r] & f_{2z+1,Q}(P) \cdot f^{zp^2}_{6z+2,Q}(P) \arrow[u]
\end{tikzcd}$$

FIGURE 4.1. Execution path for computing the α pairing on 4 processors.

4.2.2. The β Weil pairing. Next, we present the β Weil pairing, which can be computed more quickly than the α pairing, since it avoids part of the exponentiation. Consider the polynomial used in the optimal ate pairing construction given by Vercauteren [87],

$$h(x) = (6z+2) + x - x^2 + x^3.$$

Since $h(p) \equiv 0 \pmod{r}$, there exists an integer m such that $h(p) = rm$. By Theorem 3.7, we have

$$f_{p,h,R}^{-1} = f_{r,R}^{-m} \cdot f_{p,R} \cdot f_{p^2,R}^{-1} \cdot f_{p^3,R} \tag{4.5}$$

and

$$f_{p,h,R} = f_{6z+2,R} \cdot f_{-1,p^2R} \cdot \frac{\ell_{(6z+2)R,(p-p^2+p^3)R}}{v_\infty} \cdot \frac{\ell_{pR,(-p^2+p^3)R}}{v_{(p-p^2+p^3)R}} \cdot \frac{\ell_{-p^2R,p^3R}}{v_{(-p^2+p^3)R}}. \tag{4.6}$$

We use this h polynomial to construct a new variant of the Weil pairing, given by the following theorem.

Theorem 4.2. For $h(x) = (6z+2) + x - x^2 + x^3$, the map $\beta' : G_1 \times G_2 \to G_T$ defined by

$$\beta' :\ (P,Q) \mapsto w \left(\frac{f_{p,h,P}(Q)}{f_{p,h,Q}(P)}\right)^p \frac{f_{p,h,pP}(Q)}{f_{p,h,Q}(pP)} \tag{4.7}$$

is a pairing, where $w \in \mathbb{F}_p$ is some sixth root of unity.

Proof. For simplicity, multiplicative factors that are sixth roots of unity will be omitted in the proof. For $y \in \{r, p, p^2, p^3\}$, define the functions

$$\gamma_y : (P,Q) \mapsto \frac{f_{y,P}(Q)^p}{f_{y,Q}(P)^p} \cdot \frac{f_{y,pP}(Q)}{f_{y,Q}(pP)}$$

on $G_1 \times G_2$. Since

$$f_{p,h,P}^{-1} = f_{r,P}^{-m} \cdot f_{p,P} \cdot f_{p^2,P}^{-1} \cdot f_{p^3,P},$$

it follows that

$$\beta'(P,Q)^{-1} = \gamma_r(P,Q)^{-m} \cdot \gamma_p(P,Q) \cdot \gamma_{p^2}(P,Q)^{-1} \cdot \gamma_{p^3}(P,Q)$$

for all $(P,Q) \in G_1 \times G_2$. The conclusion that β' is a pairing immediately follows if it can be shown that γ_r, γ_p, γ_{p^2} and γ_{p^3} are pairings.

Now,

$$(P,Q) \mapsto \frac{f_{r,P}(Q)}{f_{r,Q}(P)}$$

and

$$(P,Q) \mapsto \frac{f_{p^2,P}(Q)}{f_{p^2,Q}(P)}$$

are, respectively, the classic Weil pairing (Theorem 2.26) and Hess's optimal Weil pairing (Theorem 3.10). It follows that γ_r and γ_{p^2} are also pairings.

Using the facts that $f_{p^2,R} = f_{p,R}^p \cdot f_{p,pR}$ (Lemma 4.1(ii)) and that $(P,Q) \mapsto f_{p,Q}(P)$ is a pairing (Theorem 2.30), the order-p Weil pairing can be written as

$$\frac{f_{p^2,P}(Q)}{f_{p^2,Q}(P)} = \frac{f_{p,P}(Q)^p}{f_{p,Q}(P)^p} \cdot \frac{f_{p,pP}(Q)}{f_{p,pQ}(P)} = \frac{f_{p,P}(Q)^p}{f_{p,Q}(P)^p} \cdot \frac{f_{p,pP}(Q)}{f_{p,Q}(pP)} = \gamma_p(P,Q),$$

and hence γ_p is a pairing.

Finally, since $(P,Q) \mapsto f_{p^2,Q}(P)$ is a pairing (Theorem 2.27) and the fact that $f_{p^3,R} = f_{p,R}^{p^2} \cdot f_{p^2,pR}$, one can check that

$$\gamma_{p^3}(P,Q) = \gamma_p(P,Q)^{p^2} \cdot \gamma_{p^2}(pP,Q).$$

Hence γ_{p^3} is a pairing. □ □

Remark 4.2. The proof of Theorem 4.2 can easily be modified for all polynomials $h(x) \in \mathbb{Z}[x]$ that satisfy $h(p) \equiv 0 \pmod r$.

Since $6 \mid p^6 - 1$ and $r \nmid (p^6-1)(p^2+1)$, the map $\beta : G_1 \times G_2 \to G_T$ defined by

$$\beta = (\beta')^{(p^6-1)(p^2+1)} : (P,Q) \mapsto \left(\left(\frac{f_{p,h,P}(Q)}{f_{p,h,Q}(P)} \right)^p \frac{f_{p,h,pP}(Q)}{f_{p,h,Q}(pP)} \right)^{(p^6-1)(p^2+1)} \tag{4.8}$$

is also a pairing. Since each extended Miller function in (4.8) is essentially a Miller function of length approximately z (see (4.6)), the β pairing is considered optimal. As was the case with the α pairing, the exponentiation by $(p^6-1)(p^2+1)$ means that the four extended Miller functions in (4.8) only need to be semi-normalized and denominator elimination can be applied. Moreover, the vertical lines $v_{(p-p^2+p^3)R}$, $v_{(-p^2+p^3)R}$, f_{-1,p^2R} and $\ell_{(6z+2)R,(p-p^2+p^3)R}$ for $R \in \{P, pP, Q\}$ in (4.6) can be ignored. Once pP has been computed, the remaining line functions $\ell_{pR,(-p^2+p^3)R}$ and ℓ_{-p^2R,p^3R} for $R \in \{P, pP, Q\}$ can be computed at very little additional cost since $p^2P = \phi(P)$, $p^3P = \phi(pP)$, and $pQ = \pi(Q)$. Furthermore, the delayed squaring technique of Section 4.1 can be employed if the extended Miller functions are divided using (4.1).

Figure 4.2 illustrates the execution path when the 4 extended Miller functions in (4.8) are computed in parallel using 4 processors. The fourth processor is the bottleneck, and thus it is desirable to accelerate the computation of pP. To this effect, we observe that

$$p \equiv 2z(p^2-2) + p^2 - 1 \pmod r,$$

whence

$$pP = 2z(p^2-2)P + p^2P - P = 2z(\phi(P) - 2P) + \phi(P) - P. \tag{4.9}$$

Thus, computing pP has roughly the same cost as zP.

$$\begin{tikzcd}
1. & f^{p}_{6z+2,P}(Q) \arrow[rr] & & f^{p}_{6z+2,P}(Q) \cdot f_{6z+2,pP}(Q) \\
2. & pP \arrow[r] & f_{6z+2,pP}(Q) \arrow[ur] & \\
3. & f^{p}_{6z+2,Q}(P) \arrow[rr] & & f^{p}_{6z+2,Q}(P) \cdot f_{6z+2,Q}(pP) \\
4. & pP \arrow[r] & f_{6z+2,Q}(pP) \arrow[ur] &
\end{tikzcd}$$

FIGURE 4.2. Execution path for computing the β pairing on 4 processors.

4.3. Parallel implementation of the BN pairings

The parallelization approaches described in Section 4.1 and Section 4.2 were implemented on top of the state-of-the-art implementation of an optimal ate pairing at the 128-bit security level described in [2]. The underlying elliptic curve is a BN curve with parameter $z = -(2^{62} + 2^{55} + 1)$ [70].

Let N denote the number of available processors on the target platform. To select parameters $(w_{N-1}, \dots, w_2, w_1, w_0)$ that split the Miller loop, we employ the load balancing scheme suggested in [3] with fine granularity, taking into account the relative cost of inversions, multiplications, squarings, additions and modular reductions on the target platform. The following parameters w_i for splitting the Miller loop were selected for the optimal ate pairing:

without delayed squaring	
2 cores	30, 0
4 cores	46, 28, 12, 0
8 cores	54, 42, 32, 24, 16, 9, 4, 0
with delayed squaring	
2 cores	35, 0
4 cores	52, 37, 19, 0
8 cores	61, 56, 49, 42, 33, 23, 12, 0

With the optimal parameters determined, elementary operation counting makes it possible to estimate the performance improvement of the corresponding implementation. Figure 4.3 presents the estimated speedups for the parallelization approaches discussed in this work in comparison with the optimal ate serial implementation of [2]. Notice how the performance of the α and β Weil pairings scales better with the number of processing cores. Scaling stills suffers from the same saturation effect experienced by the ate pairing variants, but at a higher number of cores.

The parallel implementation was realized on platforms — a 2-core Intel Core i5 Westmere 540M 32nm 2.53GHz machine ("Platform 1") and a 4-core Intel Core i7 Sandy Bridge 2630QM 32nm 2.0GHz machine ("Platform 2"), using GCC v4.6.0 as compiler with optimization flags `-O3 -funroll-loops`. Parallel sections were implemented through the compiler's native OpenMP support. The same split in the Miller loop was used in both

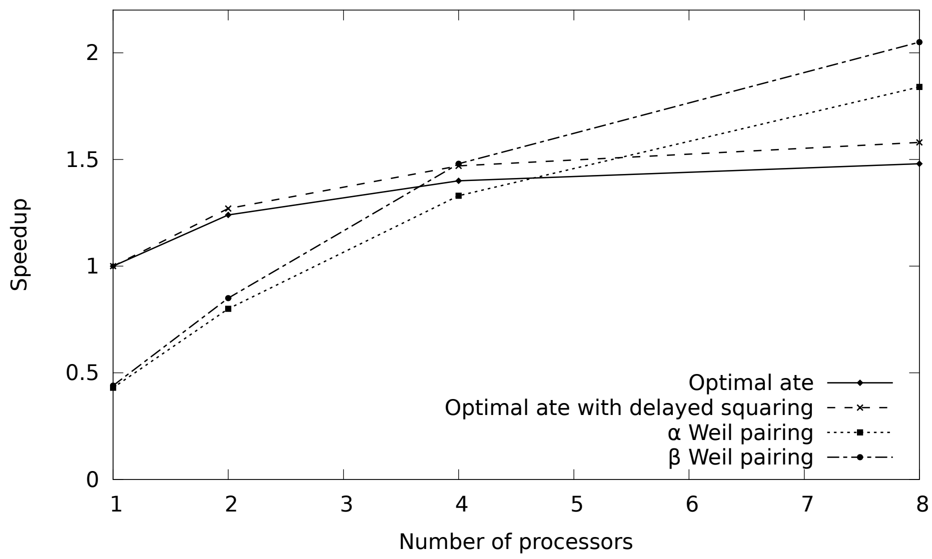

FIGURE 4.3. Estimated speedups for parallelization techniques for BN pairings. Speedups are computed in relation to a serial implementation of the optimal ate pairing.

machines, as they have similar field operation costs. Table 4.1 presents the experimental results, including both the speedups estimated by operation counting and actual timings. For the optimal ate pairing, the table confirms that delayed squaring yields a slightly better scaling, and that the increase in overall speedup starts to stagnate at eight cores. Even with these obstacles, the pairing latency of the optimal ate pairing is reduced by 18-20% when using two processor cores, a significant improvement over the 10% reported as a preliminary result for the R-ate pairing in [3]. The measured timings are compatible with the estimated speedups — the differences are due to the parallelization overheads that are not accounted for in the model for speedup estimation. The gap between the estimated and measured timings increases with the number of cores due to the linear increase in overhead.

The performance of the β Weil pairing is generally superior to the α Weil pairing due to the difference in the cost of computing zP in the former and an exponentiation by z in the cyclotomic subgroup in the latter (see Figures 4.1 and 4.2). It is important to observe that since the β pairing is tailored for parallel execution, any future improvements in the parallelization of the Miller loop in the optimal ate variants can be directly applied to the β pairing. In the columns of Table 4.1 marked with (*), we present estimates for machines with higher numbers of cores. These estimates were obtained by running multiples threads per core and then measuring the cost of the most expensive thread. This serves as an accurate prediction of performance scaling in future machines, assuming

	Number of threads			
Estimated speedup	**1**	**2**	**4**	**8**
Optimal ate	1.00	1.24	1.40	1.48
Optimal ate with delayed squaring	1.00	1.27	1.47	1.58
α Weil	0.43	0.80	1.33	1.84
β Weil	0.44	0.86	1.48	2.05
Platform 1: Intel Core i5 Nehalem 32nm	**1**	**2**	**4***	**8***
Optimal ate – latency	2038	1682	1513	1453
Optimal ate – speedup	1.00	1.21	1.35	1.40
Optimal ate with delayed squaring – latency	–	1655	1435	1389
Optimal ate with delayed squaring – speedup	–	1.23	1.42	1.47
α Weil – latency	–	–	1590	1214
α Weil – speedup	–	–	1.28	1.68
β Weil – latency	–	–	1481	1104
β Weil – speedup	–	–	1.38	1.84
Platform 2: Intel Core i7 Sandy Bridge 32nm	**1**	**2**	**4**	**8***
Optimal ate – latency	1562	1287	1137	1107
Optimal ate – speedup	1.00	1.21	1.37	1.41
Optimal ate with delayed squaring – latency	–	1260	1080	1056
Optimal ate with delayed squaring – speedup	–	1.24	1.45	1.48
α Weil – latency	–	–	1272	936
α Weil – speedup	–	–	1.23	1.67
β Weil – latency	–	–	1104	840
β Weil – speedup	–	–	1.41	1.86

TABLE 4.1. Experimental results for serial/parallel executions of BN pairings. Times are presented in thousands of clock cycles and the speedups are computed as the ratio of the execution time of serial and parallel implementations. The dashes represent data points where there is no expected improvement over the serial implementation. The columns marked with (*) present estimates based on per-thread data. Timings were obtained with the Turbo Boost feature turned off, and therefore are compatible with the timings in Table 4 of the extended version of [2].

that critical platform characteristics such as the memory organization and multi-threading overhead will not change dramatically.

CHAPTER 5

The Final Exponentiation

Efficient realizations of the Tate pairing have been intensively pursued in recent years. Using different strategies, that research effort has produced several remarkable algorithm improvements that include: construction of pairing-friendly elliptic curves with prescribed embedding degree [**10**, **31**, **63**], decreases of the Miller loop length [**7**, **42**, **43**, **87**], and reductions in the associated towering field arithmetic costs [**29**, **39**, **48**, **50**]. With the increase in efficiency of the Miller loop calculation, the final exponentiation step has become more of a computational bottleneck. Several research works have reported more refined methods for computing the final exponentiation on pairings defined over ordinary elliptic curves [**29**, **40**, **84**]. In particular, the results by Scott *et al.* [**84**] represent the current state-of-the-art in this topic, as can be verified from the fact that most recent implementations of pairings (see for example [**2**, **14**]) have obtained significant accelerations by computing the final exponentiation according to the vectorial addition chain based method described in that work.

In this chapter, we offer improvements in the final exponentiation step of pairing computation. We draw on the methods that Vercauteren (Section 3.1.2) employed to reduce the cost of the Miller function. Our results for the final exponentiation reduce the cost by a fixed number of operations in several curves, a modest but measurable improvement. Nonetheless, the techniques we use can be applied to increase the speed of hashing as well (see Chapter 6), saving a fixed number of point additions and doublings for some curves.

5.1. A lattice-based method for efficient final exponentiation

The exponent $e = (p^k - 1)/r$ in the final exponentiation of the Tate pairing can be broken into two parts by

$$(p^k - 1)/r = [(p^k - 1)/\Phi_k(p)] \cdot [\Phi_k(p)/r],$$

where $\Phi_k(z)$ denotes the k-th cyclotomic polynomial [**50**]. Computing the map

$$f \mapsto f^{(p^k-1)/\Phi_k(p)}$$

is relatively inexpensive, costing only a few multiplications, inversions, and very cheap p-th powerings in $\mathbb{F}_{p^k}$. Raising to the power $d = \Phi_k(p)/r$ is considered more difficult.

Observing that p-th powering is much less expensive than multiplication, Scott *et al.* [84] give a systematic method for reducing the expense of exponentiating by d. They showed that by writing $d = \Phi_k(p)/r$ in base p as

$$d = d_0 + d_1 p + \cdots + d_{\varphi(k)-1} p^{\varphi(k)-1},$$

one can find short vectorial addition chains to compute $f \mapsto f^d$ much more efficiently than the naive method. For parameterized curves, more concrete results can be given. For instance, BN curves are constructed over a prime field $\mathbb{F}_p$, where p is a large prime number that can be parameterized as a fourth-degree polynomial $p = p(z)$, $z \in \mathbb{Z}$. The result of Scott *et al.* gives an algorithm to compute $f \mapsto f^d$, by calculating three intermediate exponentiations, namely,

$$\begin{aligned} & f^z \\ & f^{z^2} = (f^z)^z, \\ & f^{z^3} = (f^{z^2})^z, \end{aligned}$$

along with a short sequence of products. By choosing the parameter $z \in \mathbb{Z}$ to have low hamming weight, the total cost of computing $f \mapsto f^d$ is $\frac{3}{4} \log p$ field squarings plus a small fixed number of field multiplications and squarings.

Using the fact that a fixed power of a pairing is also a pairing, it suffices to raise to the power of any multiple d' of d, with r not dividing d'. Based on this observation, we present a lattice-based method for determining d' such that $f \mapsto f^{d'}$ can be computed at least as efficiently as $f \mapsto f^d$. For BN and several other curves, explicit d' polynomials yielding more efficient final exponentiation computations are reported. However, it is noted that the main bottleneck remains, namely the exponentiation by powers of z.

In the case of parameterized curves, the key to finding suitable polynomials d' is to consider $\mathbb{Q}[z]$-multiples of $d(z)$. Specifically, we consider $\mathbb{Q}$-linear combinations of $d(z)$, $zd(z)$, ..., $z^{\deg r-1}d(z)$. To see why this set of multiples of $d(z)$ suffices, consider $f \in \mathbb{F}_{p^k}$ with order dividing $\Phi_k(p)$. Since $r(z)d(z) = \Phi_k(p)$, it follows that $f^{r(z)d(z)} = 1$ and so $f^{z^{\deg r} d(z)}$ is the product of $\mathbb{Q}$-powers of f, $f^{zd(z)}$, ..., $f^{z^{\deg r-1}d(z)}$.

Now, consider an arbitrary $\mathbb{Q}$-linear combination $d'(z)$ of the elements $d(z)$, $zd(z)$, ..., $z^{\deg r-1}d(z)$. Following the method of Scott *et al.* [84], $d'(z)$ can be written in base $p(z)$ as

$$d'(z) = d'_0(z) + d'_1(z)p(z) + \cdots + d'_{\varphi(k)-1}(z)p(z)^{\varphi(k)-1},$$

where each d'_i has degree less than the degree of p. Set

$$d'_i = d_{i,0} + zd_{i,1} + \cdots + z^{\deg p-1} d_{i,\deg p-1}$$

and assume that $d_{i,j} \in \mathbb{Z}$ for $0 \le i \le \varphi(k) - 1$, $0 \le j \le \deg(p(z)) - 1$. Then $f^{d'(z)}$ can be computed in two steps as we explain next.

First, the exponentiations f^z, ..., $f^{z^{\deg p-1}}$ are performed. From these intermediate exponentiations, terms of the form $f^{z^j p^i}$ can be easily calculated. Second, a vectorial addition chain containing the $d_{i,j}$ is found. This allows to compute $f^{d'(z)}$ from terms of the form $f^{z^j p^i}$ using the work of Olivos [68]. The advantage of allowing multiples of $d(z)$ for this computation is to provide more flexibility in the choices of the exponents $d'(z) = \sum d_{i,j} z^j p^i$ with $d_{i,j} \in \mathbb{Z}$, that can potentially yield shorter addition chains, which in turn means a more-efficient final exponentiation calculation. However the savings are necessarily modest, since as in the method of Scott *et al.* [84], the main expense in this exponentiation process comes from computing the terms $f^z, \ldots, f^{z^{\deg p-1}}$.

In order to find efficient polynomials $d'(z)$, let us construct a rational matrix M' with dimensions $\deg r \times \varphi(k) \deg p$ such that

$$\begin{bmatrix} d(z) \\ zd(z) \\ \vdots \\ z^{\deg r-1} d(z) \end{bmatrix} = M' \left(\begin{bmatrix} 1 \\ p(z) \\ \vdots \\ p(z)^{\varphi(k)-1} \end{bmatrix} \otimes \begin{bmatrix} 1 \\ z \\ \vdots \\ z^{\deg p-1} \end{bmatrix} \right).$$

Here $\otimes$ denotes the Kronecker product and the $(i, u + v \deg p)$-th entry of M' is $d_{u,v}$, where

$$z^{i-1} d(z) = \sum d_{u,v} z^{v-1} p^{u-1}.$$

Elements in the rational lattice formed by the matrix M' correspond to $\mathbb{Q}$-linear combinations $d'(z)$ of $d(z)$, $zd(z)$, $\ldots$, $z^{\deg r-1} d(z)$. Short vectorial addition chains can be produced from the elements of this lattice with small integer entries. The *LLL algorithm* of Lenstra, Lenstra, and Lovasz [53] produces an integer basis of an integer matrix with small coefficients. Let us consider the integer matrix M constructed from M' as the unique matrix whose rows are multiples of the rows of M' such that the entries of M are integers, and the greatest common divisor of the set of entries is 1. Next, the LLL algorithm is applied to M to obtain an integer basis for M with small entries. Finally, small integer linear combinations of these basis elements are examined with the hope of finding short addition chains. It is worth mentioning that even if these results do not yield an advantage over the results of Scott *et al.* [84], since the lattice contains an element corresponding to $d(z)$, the method described in this section includes the results of that work.

In the next section, the main mechanics of our method are explained by applying it to the computation of the final exponentiation step of several pairing-friendly families of elliptic curves.

5.2. Exponentiation examples

5.2.1. BN curves. BN curves have embedding degree $k = 12$ and are parameterized by z such that

$$r = r(z) = 36z^4 + 36z^3 + 18z^2 + 6z + 1,$$
$$p = p(z) = 36z^4 + 36z^3 + 24z^2 + 6z + 1$$

are both prime.

The value $d = \Phi_k(p)/r = (p^4 - p^2 + 1)/r$ can be expressed as the polynomial

$$\begin{aligned} d = d(z) &= 46656z^{12} + 139968z^{11} + 241056z^{10} + 272160z^9 \\ &+ 225504z^8 + 138672z^7 + 65448z^6 + 23112z^5 \\ &+ 6264z^4 + 1188z^3 + 174z^2 + 6z + 1. \end{aligned}$$

At first glance, it appears that exponentiations by multiples of large powers of z are required. However, following the work of Scott *et al.* [84], d can be written in base p such that the degrees of the coefficients are at most 3. In particular,

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

Scott *et al.* [84] applied the work of Olivos [68] to compute the map $f \mapsto f^d$ using vectorial addition chains. From the above representation for d, vectorial addition chains can be used to compute $f \mapsto f^d$ using 3 exponentiations by z, 13 multiplications, and 4 squarings.

For the method described in Section 5.1, consider multiples of d represented in the base p with coefficients in $\mathbb{Q}[z]/(p(z))$.

A 4×16 integer matrix M is found such that

$$\begin{bmatrix} d(z) \\ zd(z) \\ 6z^2d(z) \\ 6z^3d(z) \end{bmatrix} = M \left(\begin{bmatrix} 1 \\ p(z) \\ p(z)^2 \\ p(z)^3 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ z \\ z^2 \\ z^3 \end{bmatrix} \right).$$

The first row in M corresponds to the final exponentiation given by Scott *et al.* [84]. Any non-trivial integer linear combination of the rows corresponds to an exponentiation. For computational efficiency, a linear combination with coefficients as small as possible is desired.

None of the basis vectors returned by the LLL algorithm has an advantage over [84]. However, if small integer linear combinations of the short vectors returned by the LLL algorithm are considered, a multiple of d which corresponds to a shorter addition chain could potentially be found. A brute force search of linear combinations of the LLL basis yields 18 non-zero vectors with maximal entry 12. Among these vectors we consider the vector

$$(1, 6, 12, 12, 0, 4, 6, 12, 0, 6, 6, 12, -1, 4, 6, 12),$$

which corresponds to the multiple

$$d'(z) = \lambda_0 + \lambda_1 p + \lambda_2 p^2 + \lambda_3 p^3 = 2z(6z^2 + 3z + 1)d(z),$$

where

$$\begin{aligned}
\lambda_0(z) &= 1 + 6z + 12z^2 + 12z^3\\
\lambda_1(z) &= 4z + 6z^2 + 12z^3\\
\lambda_2(z) &= 6z + 6z^2 + 12z^3\\
\lambda_3(z) &= -1 + 4z + 6z^2 + 12z^3.
\end{aligned}$$

The final exponentiation which results can be computed more efficiently without using addition chains.

First, the following exponentiations are computed

$$f \mapsto f^z \mapsto f^{2z} \mapsto f^{4z} \mapsto f^{6z} \mapsto f^{6z^2} \mapsto f^{12z^2} \mapsto f^{12z^3}$$

which requires 3 exponentiations by z, 3 squarings, and 1 multiplication. The terms $a = f^{12z^3} \cdot f^{6z^2} \cdot f^{6z}$ and $b = a \cdot (f^{2z})^{-1}$ can be computed using 3 multiplications. Finally, the result $f^{d'}$ is obtained as

$$[a \cdot f^{6z^2} \cdot f] \cdot [b]^p \cdot [a]^{p^2} \cdot [b \cdot f^{-1}]^{p^3}$$

which requires 6 multiplications.

In total, our method requires 3 exponentiations by z, 3 squarings, and 10 multiplications. [1]

5.2.2. Freeman curves. Freeman curves [30] have embedding degree $k = 10$ and are parameterized by z such that

$$\begin{aligned}
r = r(z) &= 25z^4 + 25z^3 + 15z^2 + 5z + 1,\\
p = p(z) &= 25z^4 + 25z^3 + 25z^2 + 10z + 3
\end{aligned}$$

are both prime. For $d = \Phi_{10}(p)/r = (p^4 - p^3 + p^2 - p + 1)/r$, let us consider a 4×16 integer matrix M such that

$$\begin{bmatrix} d(z) \\ zd(z) \\ 5z^2 d(z) \\ 5z^3 d(z) \end{bmatrix} = M \left(\begin{bmatrix} 1 \\ p(z) \\ p(z)^2 \\ p(z)^3 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ z \\ z^2 \\ z^3 \end{bmatrix} \right).$$

In the lattice spanned by the rows of M, there exist two short vectors,

$$\pm(1, -2, 0, -5, -1, -4, -5, -5, 1, 3, 5, 5, 2, 5, 5, 5).$$

[1] We ignore the relatively inexpensive p-power Frobenius maps. Since the embedding degree k is even, we have $f^{-1} = f^{p^{k/2}}$ for all f in the cyclotomic subgroup of $\mathbb{F}_{p^k}^*$. That is, inversion can be done using a p-power Frobenius. Hence, we ignore inversions as well.

Both of these vectors have maximal absolute coefficient 5. Consider the vector corresponding to the multiple

$$d'(z) = (5z^3 + 5z^2 + 3z + 1)d(z) = \lambda_0 + \lambda_1 p + \lambda_2 p^2 + \lambda_3 p^3,$$

where

$$\begin{aligned}\lambda_0(z) &= 1 - 2z - 5z^3,\\ \lambda_1(z) &= -1 - 4z - 5z^2 - 5z^3,\\ \lambda_2(z) &= 1 + 3z + 5z^2 + 5z^3,\\ \lambda_3(z) &= 2 + 5z + 5z^2 + 5z^3.\end{aligned}$$

Now, the map $f \mapsto f^{d'}$ can be computed as

$$f \mapsto f^z \mapsto f^{2z} \mapsto f^{4z} \mapsto f^{5z} \mapsto f^{5z^2} \mapsto f^{5z^3},$$

followed by

$$\begin{aligned} A &= f^{5z^3} \cdot f^{2z}, & B &= A \cdot f^{5z^2},\\ C &= f^{2z} \cdot f, & D &= B \cdot f^z \cdot f,\end{aligned}$$

and finally

$$f^{d'} = [A^{-1} \cdot f] \cdot [B^{-1} \cdot C^{-1}]^p \cdot [D]^{p^2} \cdot [C \cdot D]^{p^3},$$

requiring a total of 12 multiplications, 2 squarings, and 3 exponentiations by z.

5.2.3. KSS8 curves. KSS8 curves [47] have embedding degree $k = 8$ and are parameterized by z such that

$$\begin{aligned} r = r(z) &= \frac{1}{450}(z^4 - 8z^2 + 25),\\ p = p(z) &= \frac{1}{180}(z^6 + 2z^5 - 3z^4 + 8z^3 - 15z^2 - 82z + 125)\end{aligned}$$

are both prime. For $d = \Phi_k(p)/r$, we compute an integer matrix M such that

$$\begin{bmatrix} 6d(z) \\ (6/5)zd(z) \\ (6/5)z^2 d(z) \\ (6/5)z^3 d(z) \end{bmatrix} = M \left(\begin{bmatrix} 1 \\ p(z) \\ p(z)^2 \\ p(z)^3 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ z \\ z^2 \\ z^3 \\ z^4 \\ z^5 \end{bmatrix} \right).$$

Note that since z needs to be chosen as a multiple of 5, the rows of M correspond to integer multiples of $d(z)$. We obtain a short vector corresponding to the multiple

$$d'(z) = \frac{6z}{5} d(z) = \lambda_0 + \lambda_1 p + \lambda_2 p^2 + \lambda_3 p^3$$

of $d(z)$, where

$$\begin{aligned}
\lambda_0(z) &= 2z^4 + 4z^3 + 5z^2 + 38z - 25 \\
\lambda_1(z) &= -z^5 - 2z^4 - z^3 - 16z^2 + 20z + 36 \\
\lambda_2(z) &= z^4 + 2z^3 - 5z^2 + 4z - 50 \\
\lambda_4(z) &= 3z^3 + 6z^2 + 15z + 72.
\end{aligned}$$

We use addition chains to compute $f^{d'}$. First, write $f^{d'}$ as

$$f^{d'} = y_0^1 y_1^2 y_2^3 y_3^4 y_4^5 y_5^6 y_6^{15} y_7^{16} y_8^{20} y_9^{25} y_{10}^{36} y_{11}^{38} y_{12}^{50} y_{13}^{72}$$

and compute the y_i. The y_i can be computed from f, f^z, ..., f^{z^5} using only multiplications and Frobenius maps.

Next, we find an addition chain containing all the powers of the y_i. With the inclusion of the element 10, we obtain

$$\{1, 2, 3, 4, 5, 6, \underline{10}, 15, 16, 20, 25, 36, 38, 50, 72\}.$$

The work of Olivos gives an efficient method for producing a vectorial addition chain from an addition chain and states the computational expense of computing the final result $f^{d'}$ from the y_i.

The computation of the y_i requires 5 exponentiations by z and 6 multiplications. The addition chain requires 7 multiplications and 7 squarings. The conversion to a vectorial addition chain requires 13 multiplications. In total, we require 5 exponentiations by z, 26 multiplications, and 7 squarings to compute the map $f \mapsto f^{d'}$.

5.2.4. KSS18 curves. KSS18 curves [47] have embedding degree $k = 18$ and a twist of order $d = 6$. These curves are parameterized by z such that

$$\begin{aligned}
r = r(z) &= \frac{1}{343}(z^6 + 37z^3 + 343), \\
p = p(z) &= \frac{1}{21}(z^8 + 5z^7 + 7z^6 + 37z^5 + 188z^4 \\
&\qquad + 259z^3 + 343z^2 + 1763z + 2401)
\end{aligned}$$

are both prime. For $d = \Phi_k(p)/r$, we compute an integer matrix M such that

$$\begin{bmatrix} 3d(z) \\ (3/7)zd(z) \\ (3/49)z^2d(z) \\ (3/49)z^3d(z) \\ (3/49)z^4d(z) \\ (3/49)z^5d(z) \end{bmatrix} = M \left(\begin{bmatrix} 1 \\ p(z) \\ p(z)^2 \\ p(z)^3 \\ p(z)^4 \\ p(z)^5 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ z \\ z^2 \\ z^3 \\ z^4 \\ z^5 \\ z^6 \\ z^7 \end{bmatrix} \right).$$

Since 7 divides z, the rows of M correspond to integer multiples of $d(z)$. We find a short vector corresponding to the multiple

$$d'(z) = \frac{3z^2}{49} d(z) = \lambda_0 + \lambda_1 p + \lambda_2 p^2 + \lambda_3 p^3 + \lambda_4 p^4 + \lambda_5 p^5$$

of $d(z)$, where

$$\begin{aligned}
\lambda_0(z) &= z^6 + 5z^5 + 7z^4 + 21z^3 + 108z^2 + 147z, \\
\lambda_1(z) &= -5z^5 - 25z^4 - 35z^3 - 98z^2 - 505z - 686, \\
\lambda_2(z) &= -z^7 - 5z^6 - 7z^5 - 19z^4 - 98z^3 - 133z^2 + 6, \\
\lambda_3(z) &= 2z^6 + 10z^5 + 14z^4 + 35z^3 + 181z^2 + 245z, \\
\lambda_4(z) &= -3z^5 - 15z^4 - 21z^3 - 49z^2 - 254z - 343, \\
\lambda_5(z) &= z^4 + 5z^3 + 7z^2 + 3.
\end{aligned}$$

Proceeding as in the KSS8 example, we construct an addition chain

$$\{1, 2, 3, 5, 6, 7, 10, 14, 15, 19, 21, 25, 35, \underline{38}, 49, \underline{73}, \\ 98, 108, 133, 147, 181, 245, 254, 343, \underline{490}, 505, 686\}.$$

Once again, applying Olivos' method for computing a short vectorial addition chain, we can compute the map $f \mapsto f^d$ using 7 exponentiations by z, 52 multiplications, and 8 squarings.

5.2.5. A comparison with Scott et al. In Table 5.1, we compare our results against those given by Scott *et al.* [84]. Although operation counts are given for only the vectorial addition portion of the exponentiation, the total cost can easily be computed from their work. The operation counts are given for field multiplications and squarings only, since the number of exponentiations by z is fixed for each curve and computing p-th power maps is comparatively inexpensive.

Curve	Scott *et al.*	This work
BN	13M 4S	10M 3S
Freeman	14M 2S	12M 2S
KSS8	31M 6S	26M 7S
KSS18	62M 14S	52M 8S

TABLE 5.1. A comparison of our final exponentiation method with the method of Scott *et al.* [84]. 'M' denotes a multiplication and 'S' denotes a squaring. Both methods require the same number of exponentiations by z, determined by the curve.

For example, let us consider the case of BN curves parameterized with $z = -2^{62} - 2^{54} + 2^{44}$, which yields a 127-bit security level [14]. Further, assume that the relative cost of a field multiplication compared to a cyclotomic squaring in $\mathbb{F}_{p^k}$ is given as $M \approx 4.5S$ [2, 48]. Then the total cost to perform the exponentiations $f^z, (f^z)^z, (f^{z^2})^z$, is of around $3 \cdot \lfloor \log_2 z \rfloor = 183$ cyclotomic squarings. Using the results reported in Table 5.1, this gives an approximate cost for the hard part of the final exponentiation of $187S+13M \approx 245S$ for the method of Scott *et al.* and $186S + 10M \approx 231S$ using our method.

CHAPTER 6

Hashing to Elliptic Curve Points

Some pairing-based protocols, such as Scott's identity-based key agreement scheme (Section 1.3.2) require the computation of random points in G_1 and G_2. In the literature, this problem is referred to as *hashing to* G_1 and *hashing to* G_2, respectively. The group G_1 is defined as $E(\mathbb{F}_p)[r]$. Hashing to G_1 is normally seen as a straightforward task, whereas hashing to G_2 is considered more challenging.

The customary method for representing G_2 is as the order-r subgroup of $\tilde{E}(\mathbb{F}_{p^{k/d}})$, where $\tilde{E}$ is the degree-d twist of E over $\mathbb{F}_{p^{k/d}}$ with $r \mid \#\tilde{E}(\mathbb{F}_{p^{k/d}})$ (Section 2.4). Hashing to G_2 can be accomplished by finding a random point $Q \in \tilde{E}(\mathbb{F}_{p^{k/d}})$ followed by a multiplication by $c = \#\tilde{E}(\mathbb{F}_{p^{k/d}})/r$. The main difficulty of this hashing is that c is normally a relatively large scalar (for example, larger than p). Galbraith and Scott [**37**] reduce the computational cost of this task by means of an endomorphism of $\tilde{E}$. This idea was further exploited by Scott *et al.* [**83**], where explicit formulae for hashing to G_2 were given for several pairing-friendly curves.

In this chapter, we offer improvements in hashing to G_2. We draw on the methods that Vercauteren employed to reduce the cost of the Miller function (Section 3.1.2). Our framework for fast hashing produces hashing algorithms that are much faster than the currently best-known methods. For example, we estimate that for BN curves at the 128-bit security level, our results yield a hashing algorithm that is at least two times faster than the previous fastest known algorithm. For higher embedding degree curves, the results can be more dramatic.

6.1. A lattice-based method for hashing to G_2

Let E be an elliptic curve over $\mathbb{F}_p$ with r, a large prime divisor of $n = \#E(\mathbb{F}_p)$, and let $k > 1$ be the embedding degree of E. Let q be an arbitrary power of p. Recall that an elliptic curve $\tilde{E}$ defined over $\mathbb{F}_q$ is said to be a *degree-d twist of E over* $\mathbb{F}_q$, if d is the smallest integer such that $\tilde{E}$ and E are isomorphic over $\mathbb{F}_{q^d}$. If p has characteristic greater than 3, the only possible degrees of twists are those integers d which divide either 4 or 6. Since our examples deal only with curves where the degree of the twist divides the embedding degree k, we assume that d divides k and set $q = p^{k/d}$ for the remainder of this chapter. However, with some modifications, the preliminary discussion and results apply to curves where d does not divide k.

Hess *et al.* [43] showed that there exists a unique non-trivial twist $\tilde{E}$ of E over $\mathbb{F}_q$ such that r divides $\#\tilde{E}(\mathbb{F}_q)$. If $d = 2$, then $\#\tilde{E}(\mathbb{F}_q) = q + 1 + \hat{t}$, where $\hat{t}$ is the trace of the q-power Frobenius of E. In fact, the order of any twist can be found by first determining the trace $\hat{t}$ of the q-power Frobenius of E from the trace t of the p-power Frobenius of E via the Weil Theorem and then using a result given by Hess *et al.* [43].

The trace t_m of the p^m-power Frobenius of E for an arbitrary m can be determined using the recursion [59]

$$\begin{aligned} t_0 &= 2, \\ t_1 &= t, \\ t_{i+1} &= t \cdot t_i - p \cdot t_{i-1}, && \text{for all } i > 1. \end{aligned}$$

After computing the trace $\hat{t}$ of the q-power Frobenius of E, the possible values for the trace $\tilde{t}$ of the q-power Frobenius of $\tilde{E}$ over $\mathbb{F}_q$ can be determined using Table 6.1 [43], where D is the discriminant of E and $\hat{f}$ satisfies $\hat{t}^2 - 4q = D\hat{f}^2$.

d	2	3	4	6
$\tilde{t}$	$-\hat{t}$	$(\pm 3\hat{f} - \hat{t})/2$	$\pm\hat{f}$	$(\pm 3\hat{f} + \hat{t})/2$

TABLE 6.1. Possible values for the trace $\tilde{t}$ of the q-power Frobenius of a degree-d twist $\tilde{E}$ of E.

The group G_2 can be represented as $\tilde{E}(\mathbb{F}_q)[r]$. In order to hash to G_2, it suffices to hash to a random point $Q \in \tilde{E}(\mathbb{F}_q)$ followed by a multiplication by the cofactor $c = \#\tilde{E}(\mathbb{F}_q)/r$, to obtain the element $cQ \in \tilde{E}(\mathbb{F}_q)[r]$. Let $\phi\colon \tilde{E} \to E$ be an efficiently-computable isomorphism defined over $\mathbb{F}_{q^d}$ and let π be the p-th power Frobenius on E. Scott *et al.* [83] observed that the endomorphism

$$\psi = \phi^{-1} \circ \pi \circ \phi$$

can be used to speed up the computation of $Q \mapsto cQ$. The endomorphism ψ satisfies

$$\psi^2 P - t\psi P + pP = \infty \tag{6.1}$$

for all $P \in \tilde{E}(\mathbb{F}_q)$ [35, Theorem 1]. The cofactor c can be written as a polynomial in p with coefficients less than p. Scott *et al.* use this representation of c and reduce using (6.1) so that c is expressed as a polynomial in ψ with coefficients less than p. For parameterized curves, the speedup in the cost of computing $Q \mapsto cQ$ can become quite dramatic. For example, MNT curves [63] have embedding degree $k = 6$ and are parameterized by z such that

$$\begin{aligned} p(z) &= z^2 + 1 \\ r(z) &= z^2 - z + 1 \end{aligned}$$

are both prime. It can be shown that

$$\begin{aligned} c(z)P &= (z^4 + z^3 + 3z^2)P = (p^2 + (z+1)p - z - 2)P \\ &= \psi(2zP) + \psi^2(2zP). \end{aligned}$$

It suffices to multiply by a multiple c' of c such that $c' \not\equiv 0 \pmod{r}$. Combining this observation with a new method of representing c in base ψ, we prove the following theorem.

Theorem 6.1. Suppose that $\tilde{E}(\mathbb{F}_q)$ is cyclic and $p \equiv 1 \pmod{d}$. Then there exists a polynomial $h(x) = h_0 + h_1 x + \cdots + h_{\varphi(k)-1}x^{\varphi(k)-1} \in \mathbb{Z}[x]$ such that $h(\psi)P$ is a fixed multiple of cP for all $P \in \tilde{E}(\mathbb{F}_q)$ and $|h_i|^{\varphi(k)} \leq \#\tilde{E}(\mathbb{F}_q)/r$ for all i.

The proof of Theorem 6.1 is divided into two parts. We first prove a technical lemma and then show how the polynomial h can be obtained using an integer-lattice technique. Let f, $\tilde{f}$ be such that $t^2 - 4p = Df^2$ and $\tilde{t}^2 - 4q = D\tilde{f}^2$, where D is the discriminant. It also holds that $n + t = p + 1$ and $\tilde{n} + \tilde{t} = q + 1$, where $\tilde{n} = \#\tilde{E}(\mathbb{F}_q)$.

Recall that the endomorphism $\psi\colon \tilde{E} \to \tilde{E}$ is defined over $\mathbb{F}_{q^d}$. In the following lemma, it is proved that ψ fixes $\tilde{E}(\mathbb{F}_q)$ as a set.

Lemma 6.2. If $p \equiv 1 \pmod{d}$, then $\psi P \in \tilde{E}(\mathbb{F}_q)$ for all $P \in \tilde{E}(\mathbb{F}_q)$.

Proof. From the work of Hess *et al.* [43] we have that the twist is defined by first selecting $\gamma \in \mathbb{F}_{q^d}$ such that $\gamma^d \in \mathbb{F}_q$. The map ϕ is then defined by $\phi(x,y) = (\gamma^2 x, \gamma^3 y)$ and hence ψ is defined by $\psi(x,y) = (\gamma^{2(p-1)}x^p, \gamma^{3(p-1)}y^p)$. Now, $\gamma^d \in \mathbb{F}_q$ and $p - 1 \equiv 0 \pmod{d}$ yield $\gamma^{p-1} \in \mathbb{F}_q$, which in turn implies that $\psi(x,y) \in \tilde{E}(\mathbb{F}_q)$ for $(x,y) \in \tilde{E}(\mathbb{F}_q)$. □

The following lemma illustrates the effect of ψ on elements in $\tilde{E}(\mathbb{F}_q)$.

Lemma 6.3. If $p \equiv 1 \pmod{d}$, $\gcd(\tilde{f}, \tilde{n}) = 1$, and $\tilde{E}(\mathbb{F}_q)$ is a cyclic group, then $\psi P = aP$ for all $P \in \tilde{E}(\mathbb{F}_q)$, where a is one of $(t + f(\tilde{t} - 2)/\tilde{f})/2$, $(t - f(\tilde{t} - 2)/\tilde{f})/2$.

Proof. Since $\tilde{E}(\mathbb{F}_q)$ is cyclic and ψ fixes $\tilde{E}(\mathbb{F}_q)$, there exists an integer a such that $\psi P = aP$ for all $P \in \tilde{E}(\mathbb{F}_q)$. By solving for a in (6.1) and using the fact that $t^2 - 4p = Df^2$, we obtain

$$a \equiv \frac{1}{2}(t \pm \sqrt{t^2 - 4p}) \equiv \frac{1}{2}(t \pm \sqrt{Df^2}) \equiv \frac{1}{2}(t \pm f\sqrt{D}) \pmod{\tilde{n}}.$$

Working modulo $\tilde{n}$, we observe that

$$D\tilde{f}^2 = \tilde{t}^2 - 4q = \tilde{t}^2 - 4\tilde{t} + 4 = (\tilde{t} - 2)^2$$

and so

$$\sqrt{D} \equiv \pm(\tilde{t} - 2)/\tilde{f} \pmod{\tilde{n}}.$$

Without loss of generality, let f, $\tilde{f}$ be such that $a = \frac{1}{2}(t + f\sqrt{D})$ and $\sqrt{D} \equiv (\tilde{t} - 2)/\tilde{f} \pmod{\tilde{n}}$. Then, since $P \in \tilde{E}(\mathbb{F}_q)$ has order dividing $\tilde{n}$, it follows that

$$\psi P = aP = \left(\frac{1}{2}(t + f\sqrt{D})\right) P = \left(\frac{1}{2}(t + f(\tilde{t} - 2)/\tilde{f})\right) P.$$

□

In the space of polynomials $h \in \mathbb{Z}[x]$ such that $h(a) \equiv 0 \pmod{c}$, we wish to find an h with small integer coefficients. Ignoring the small coefficient requirement for the moment, $h(x) = c$ and $h(x) = x^i - a^i$ satisfy the required condition for all integers i. Furthermore, any integer linear combination of these polynomials satisfies this condition.

Since π acting on $E(\mathbb{F}_{p^k})$ has order k and ψ is an automorphism when restricted to the cyclic group $\tilde{E}(\mathbb{F}_q)$, the automorphism ψ acting on $\tilde{E}(\mathbb{F}_q)$ has order k. Hence, the integer a satisfies $\Phi_k(a) \equiv 0 \pmod{\tilde{n}}$. Therefore, the polynomial $h(x) = x^i - a^i$ with $i \geq \varphi(k)$ can be written as an integer linear combination (modulo c) of $x - a, \ldots, x^{\varphi(k)-1} - a^{\varphi(k)-1}$. For this reason, the polynomials of higher degree are excluded in the following construction.

Notice that points in the integer lattice generated by the matrix

$$\left[\begin{array}{c|c} c & \mathbf{0} \\ \hline \mathbf{a} & I_{\varphi(k)-1} \end{array}\right],$$

where $\mathbf{a}$ is the column vector with i-th entry $-a^i$, correspond to polynomials $h \in \mathbb{Z}[z]$ such that $h(a) \equiv 0 \pmod{c}$. Consider the convex set $C \subseteq \mathbb{R}^{\varphi(k)}$ generated by all vectors of the form

$$(\pm|c|^{1/\varphi(k)}, \ldots, \pm|c|^{1/\varphi(k)}).$$

The volume of C is $2^{\varphi(k)}|c|$ and the lattice above has volume $|c|$. By Minkowski's Theorem (Theorem 3.16), the region C contains a lattice point. Hence, there exists a non-zero polynomial h with coefficients at most $|c|^{1/\varphi(k)}$ such that $h(a) \equiv 0 \pmod{c}$. This concludes the proof of Theorem 6.1. □

6.2. Hashing examples

6.2.1. BN curves.

6.2.1. BN curves. BN curves have embedding degree $k = 12$ and are parameterized by

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

where

$$\begin{aligned} t(z)^2 - 4p(z) &= -3f(z)^2 \\ r(z) + t(z) &= p(z) + 1 \\ q(z) &= p(z)^2. \end{aligned}$$

After computing the trace $\hat{t}$ of the q-power Frobenious of E, we compute $\hat{f}$ such that $4q - \hat{t} = -3\hat{f}^2$. Using $\hat{t}$ and $\hat{f}$, we find the twist $\tilde{E}(\mathbb{F}_q)$ is parameterized by

$$\begin{aligned} \tilde{n}(z) &= q(z) + 1 - (3\hat{f}(z) + \hat{t}(z))/2 \\ &= (36z^4 + 36z^3 + 18z^2 + 6z + 1)(36z^4 + 36z^3 + 30z^2 + 6z + 1) \\ \tilde{t}(z) &= 36z^4 + 1. \end{aligned}$$

Now $c(z) = p(z) + t(z) - 1$ is such that $\tilde{n}(z) = r(z)c(z)$. Using Lemma 6.3, we obtain

$$\begin{aligned} a(z) &= \frac{1}{2}(t + f(\tilde{t} - 2)/\tilde{f}) \\ &= -\frac{1}{5}(3456z^7 + 6696z^6 + 7488z^5 + 4932z^4 + 2112z^3 + 588z^2 + 106z + 6). \end{aligned}$$

As a sobriety check, note that $a(z) \equiv p(z) \pmod{r}$ and thus $\psi Q = a(z)Q = p(z)Q$ for all $Q \in \tilde{E}(\mathbb{F}_q)[r]$.

We construct the following lattice and reduce the $-a(z)^i$ entries modulo $c(z)$:

$$\left[\begin{array}{c|ccc} c(z) & 0 & 0 & 0 \\ -a(z) & 1 & 0 & 0 \\ -a(z)^2 & 0 & 1 & 0 \\ -a(z)^3 & 0 & 0 & 1 \end{array}\right] \to \left[\begin{array}{c|ccc} 36z^4 + 36z^3 + 30z^2 + 6z + 1 & 0 & 0 & 0 \\ 48/5z^3 + 6z^2 + 4z - 2/5 & 1 & 0 & 0 \\ 36/5z^3 + 6z^2 + 6z + 1/5 & 0 & 1 & 0 \\ 12z^3 + 12z^2 + 8z + 1 & 0 & 0 & 1 \end{array}\right].$$

From this lattice, we find the polynomial $h(x) = z + 3zx + zx^2 + x^3$. Working modulo $\tilde{n}(z)$, we have

$$h(a) = -(18z^3 + 12z^2 + 3z + 1)c(z)$$

and since $\gcd(18z^3 + 12z^2 + 3z + 1, r(z)) = 1$, the following map is a homomorphism of $\tilde{E}(\mathbb{F}_q)$ with image $\tilde{E}(\mathbb{F}_q)[r]$:

$$Q \mapsto zQ + \psi(3zQ) + \psi^2(zQ) + \psi^3(Q).$$

We can compute $Q \mapsto zQ \mapsto 2zQ \mapsto 3zQ$ using one doubling, one addition, and one multiply-by-z. Given Q, zQ, $3zQ$, we can compute $h(a)Q$ using three ψ-maps, and three additions. In total, we require one doubling, four additions, one multiply-by-z, and three ψ-maps. As seen in Table 6.2 on page 78, the previous fastest-known method of computing such a homomorphism costs two doublings, four additions, two multiply-by-z's, and three ψ-maps.

6.2.2. Freeman curves. Freeman curves [30] have embedding degree $k = 10$ and are parameterized by z such that

$$r = r(z) = 25z^4 + 25z^3 + 15z^2 + 5z + 1,$$
$$p = p(z) = 25z^4 + 25z^3 + 25z^2 + 10z + 3,$$

are both prime.

Since Freeman curves do not have a fixed discriminant, the algorithm given in the proof of Lemma 6.3 does not directly apply. However, we are able to apply the techniques of Scott *et al.* on $c(z)$, $xc(z)$, $z^2c(z)$, $z^3c(z)$ and then use our method from Chapter 5.

We find a short vector corresponding to the multiple $h(a) = \lambda_0 + \lambda_1 a + \lambda_2 a^2 + \lambda_3 a^3$ of c, where $\lambda = (\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4)$ is such that

$$\begin{aligned}
\lambda_0(z) &= 10z^3 + 5z^2 + 4z + 1\\
\lambda_1(z) &= -3z\\
\lambda_2(z) &= -10z^3 - 10z^2 - 8z - 3\\
\lambda_3(z) &= -5z^3 - 5z^2 - z\\
\lambda_4(z) &= -5z^3 + 2.
\end{aligned}$$

Using the addition chain $\{1, 2, 3, 4, 5, 8, 10\}$, we can compute $h(a)Q$ using fourteen additions, four doublings, three multiply-by-z's, and four ψ maps.

6.2.3. KSS8 curves. KSS8 curves [47] have embedding degree $k = 8$ and are parameterized by z such that

$$r = r(z) = \frac{1}{450}(z^4 - 8z^2 + 25)$$
$$p = p(z) = \frac{1}{180}(z^6 + 2z^5 - 3z^4 + 8z^3 - 15z^2 - 82z + 125)$$

are both prime. Set $q = p^2$. There exists a degree-4 twist $\tilde{E}(\mathbb{F}_q)$ of order

$$\tilde{n}(z) = \frac{1}{72}(z^8 + 4z^7 + 6z^6 + 36z^5 + 34z^4 - 84z^3 + 486z^2 + 620z + 193)r(z).$$

Set $c(z) = \tilde{n}(z)/r(z)$. After some work, we discover that ψ is such that $\psi Q = aQ$ for all $Q \in \tilde{E}(\mathbb{F}_q)$ where

$$\begin{aligned}
a = \frac{1}{184258800}\big(&-52523z^{11} - 174115z^{10} + 267585z^9 - 193271z^8\\
&- 325290z^7 + 15093190z^6 - 29000446z^5 - 108207518z^4\\
&+ 235138881z^3 + 284917001z^2 - 811361295z - 362511175\big).
\end{aligned}$$

As we have done previously, we find a short basis for the lattice generated by the matrix

$$\left[\begin{array}{c|ccc} c(z) & 0 & 0 & 0 \\ -a(z) & 1 & 0 & 0 \\ -a(z)^2 & 0 & 1 & 0 \\ -a(z)^3 & 0 & 0 & 1 \end{array}\right]$$

and discover a short vector corresponding to the multiple

$$h(a) = \frac{1}{75}(z^2 - 25)c(z) = \lambda_0 + \lambda_1 a + \lambda_2 a^2 + \lambda_3 a^3$$

of c such that $\lambda = (\lambda_0, \lambda_1, \lambda_2, \lambda_3) = (-z^2 - z, z - 3, 2z + 6, -2z - 4)$.

For an element $Q \in \tilde{E}(\mathbb{F}_q)$, we can compute $h(a)Q$ with the following sequence of calculations. We compute $Q \mapsto zQ \mapsto (z+1)Q \mapsto (z^2+z)Q$ and $Q \mapsto 2Q \mapsto 4Q$ which requires one addition, two doublings, and two multiply-by-z's. Then we compute

$$\begin{aligned} \lambda_0 Q &= -(z^2 + z)Q \\ \lambda_1 Q &= (z+1)Q - 4Q \\ \lambda_2 Q &= 2(z+1)Q + 4Q \\ \lambda_3 Q &= -2(z+1)Q - 2Q \end{aligned}$$

which requires three more additions and another doubling. Finally, we compute

$$h(a)Q = \lambda_0 Q + \psi(\lambda_1 Q) + \psi^2(\lambda_2 Q) + \psi^3(\lambda_3 Q)$$

which requires three more additions and three ψ maps.

In total, we require seven additions, three doublings, two multiply-by-z's, and three ψ maps to compute $Q \mapsto h(a)Q$.

6.2.4. KSS18 curves. KSS18 curves [47] have embedding degree $k = 18$ and a twist of order $d = 6$. These curves are parameterized by z such that

$$\begin{aligned} r = r(z) &= \frac{1}{343}(z^6 + 37z^3 + 343) \\ p = p(z) &= \frac{1}{21}(z^8 + 5z^7 + 7z^6 + 37z^5 + 188z^4 \\ &\qquad + 259z^3 + 343z^2 + 1763z + 2401) \end{aligned}$$

are both prime. We find that

$$\begin{aligned}
c_1(z) &= \frac{49}{3}(z^2 + 5z + 7) \\
c_2(z) &= \frac{1}{27}\big(z^{18} + 15z^{17} + 96z^{16} + 409z^{15} + 1791z^{14} + 7929z^{13} + 27539z^{12} \\
&\quad + 81660z^{11} + 256908z^{10} + 757927z^9 + 1803684z^8 \\
&\quad + 4055484z^7 + 9658007z^6 + 19465362z^5 + 30860595z^4 \\
&\quad + 50075833z^3 + 82554234z^2 + 88845918z + 40301641\big).
\end{aligned}$$

Constructing our lattice, we obtain the vector corresponding to the multiple

$$h(a) = -\frac{3}{343}z(8z^3 + 147)c(z) = \lambda_0 + \lambda_1 a + \lambda_2 a^2 + \lambda_3 a^3 + \lambda_2 a^4 + \lambda_3 a^5$$

of $c(z)$, where

$$\begin{aligned}
\lambda_0 &= 5z + 18 \\
\lambda_1 &= z^3 + 3z^2 + 1 \\
\lambda_2 &= -3z^2 - 8z \\
\lambda_3 &= 3z + 1 \\
\lambda_4 &= -z^2 - 2 \\
\lambda_5 &= z^2 + 5z.
\end{aligned}$$

We construct the addition chain $\{1, 2, 3, 5, 8, \underline{10}, 18\}$, from which we can compute $Q \mapsto h(a)Q$ using sixteen additions, two doublings, three multiply-by-z's, and five ψ maps.

6.2.5. Comparison with previous work. In Table 6.2, we compare our results to the work of Scott *et al.* [83, 82]. In the proceedings version [83] of their work, the authors assume that the identity $\Phi_k(\psi)P = \infty$ holds for all points P in $\tilde{E}(\mathbb{F}_q)$. However, there exist concrete examples showing that this identity does *not* hold for some curves. In particular, MNT and Freeman curves do not satisfy this identity in general. On the other hand, the identity $\psi^{k/2}P = -P$ is critically used in the eprint version [82] of their work. Fortunately, all curves we have discussed with the exception of the MNT curve can be explicitly shown to satisfy the identity $\psi^{k/2}P = -P$. In practice, when we fix the parameter of an MNT curves, we have found that these concrete curves also satisfy this property. It would be interesting to discover an MNT curve which does not satisfy this property. More work needs to be done to determine the structure of the twist and the action of ψ on various subgroups of the twist for such curves.

We use the eprint version [82] to represent Scott *et al.*'s operation counts on Freeman curves. We have verified that the identity $\Phi_k(\psi)P = \infty$ holds for BN, KSS8, and KSS18 curves and use the counts from the proceedings version [83] of their work for those curves

Curve	Scott *et al.*	This work
BN	4A 2D 2Z 3ψ	4A 1D 1Z 3ψ
Freeman	20A 5D 3Z 4ψ	14A 4D 3Z 4ψ
KSS8	22A 5D 5Z 2ψ	7A 3D 2Z 3ψ
KSS18	59A 5D 7Z 4ψ	16A 2D 3Z 5ψ

TABLE 6.2. A comparison of our hashing algorithm with the hashing algorithm of Scott *et al.* 'A' denotes a point addition, 'D' denotes a point doubling, 'Z' denotes a multiplication by z, and 'ψ' denotes an application of the map ψ.

in Table 6.2. Since the multiplications by z dominate the other operations, it can be seen that our hash algorithm is approximately twice as fast as that of Scott *et al.* for BN curves. For KSS8 curves we see a $\frac{5}{2}$-fold improvement, and for KSS18 curves, we see a $\frac{7}{3}$-fold improvement.

CHAPTER 7

Pairings at High Security Levels

Initial work on efficiently computable pairings [7, 32] was focused on implementing pairings at (roughly) the 80-bit security level. Koblitz and Menezes [50] highlighted the performance drawbacks of pairings at very high security levels. The subsequent discovery of BN curves [10], ideally suited for implementing pairings at the 128-bit security level, spurred a lot of research culminating in the implementation of Aranha et al. [2] that achieved speeds of under 2 million cycles for a 128-bit pairing computation on a single core of Phenom II, Core i5 and Opteron machines.

More recently, researchers have considered implementing pairings at even higher security levels. Costello, Lauter and Naehrig [28] argued that a certain family of embedding degree $k = 24$ Barreto-Lynn-Scott elliptic curves [8], henceforth called *BLS24* curves, are well-suited for implementing pairings at the 192, 224, 256, 288, and 320-bit security levels. Scott [80] implemented several pairing-based protocols using BN curves at the 128-bit security level, KSS18 curves [47] with embedding degree $k = 18$ at the 192-bit security level, and BLS24 curves at the 256-bit security level. Scott concludes that the best choice of pairing to implement a particular protocol can depend on a variety of factors including the number and complexity of non-pairing operations in the protocol, the number of pairing computations that are required, and the applicability of several optimizations including fixed-argument pairings and products of pairings [79].

In this chapter, we focus on fast implementations of a single pairing at the 192-bit security level. We chose the 192-bit level because it is the higher security level (the other is 128-bit) for public-key operations in the National Security Agency's Suite B Cryptography standard [85]. Moreover, as mentioned by Scott [80], the optimum choice of pairing-friendly curve for the 192-bit security level from the many available candidates [31] is not straightforward.

We examine a family of embedding degree $k = 12$ elliptic curves, henceforth called *BLS12 curves*, first proposed by Barreto, Lynn and Scott [8] (see also [19]). Unlike BN curves, the BLS12 curves are not ideal for the 128-bit security level since the group order $\#E(\mathbb{F}_p)$ is not prime. Nevertheless, our careful estimates and implementation results demonstrate that they outperform KSS18, BN and BLS24 curves at the 192-bit security level. We also present a general framework for deriving analogues of the β Weil pairing, presented in Chapter 4 for BN curves. This pairing is well-suited for computing a single pairing on a multi-processor machine since it avoids the relatively-costly final exponentiation that cannot be effectively parallelized and is present in all Tate-type pairings.

7.1. Selection of pairings

Table 7.1 summarizes the salient parameters of the KSS18 [47], BN [10], BLS12 [8] and BLS24 [8] families of elliptic curves. All these curves are parameterized by a positive integer z, are defined by an equation of the form $Y^2 = X^3 + b$, and have a twist of order $d = 6$. Table 7.2 lists the important parameters of the particular KSS18, BN, BLS12 and BLS24 curves that are suitable for implementing pairing-based protocols at the 192-bit security level. The requirements for this security level are that the bitlength of r be at least 384 (in order to resist Pollard's rho attack [71] on the discrete logarithm problem in G_1), and that the bitlength of p^k should be at least 7680 (in order to resist the number field sieve attack [77] on the discrete logarithm problem in $\mathbb{F}_{p^k}^*$).

KSS18 curves: $k = 18$ $p(z) = (z^8 + 5z^7 + 7z^6 + 37z^5 + 188z^4 + 259z^3 + 343z^2 + 1763z + 2401)/21$ $r(z) = (z^6 + 37z^3 + 343)/343$, $t(z) = (z^4 + 16z + 7)/7$
BN curves: $k = 12$ $p(z) = 36z^4 + 36z^3 + 24z^2 + 6z + 1$ $r(z) = 36z^4 + 36z^3 + 18z^2 + 6z + 1$, $t(z) = 6z^2 + 1$
BLS12 curves: $k = 12$ $p(z) = (z-1)^2(z^4 - z^2 + 1)/3 + z$, $r(z) = z^4 - z^2 + 1$, $t(z) = z + 1$
BLS24 curves: $k = 24$ $p(z) = (z-1)^2(z^8 - z^4 + 1)/3 + z$, $r(z) = z^8 - z^4 + 1$, $t(z) = z + 1$

TABLE 7.1. Important parameters for the KSS18, BN, BLS12 and BLS24 families.

Optimal ate pairings for KSS18 [87], BN [87], BLS12 [43] and BLS24 [43] curves are given in Table 7.3. In the table, $\ell_{S,T}$ denotes the line through points S and T.

7.1.1. The β Weil pairing. Set $k = ed$, where d is the order of the automorphism group of E. Define w_s and $w_{s,h}$ as

$$w_s(P,Q) = \left(\frac{f_{s,Q}(P)}{f_{s,P}(Q)}\right)^{p^{k/2}-1} \quad \text{and} \quad w_{s,h}(P,Q) = \left(\frac{f_{s,h,Q}(P)}{f_{s,h,P}(Q)}\right)^{p^{k/2}-1}. \tag{7.1}$$

Curve	b	k	z	$\lceil\log_2 p\rceil$	$\lceil\log_2 r\rceil$	ρ	$\lceil\log_2 q\rceil$	$\lceil\log_2 p^k\rceil$
KSS18	2	18	$-2^{64}-2^{51}+2^{46}+2^{12}$	508	376	1.35	1523	9137
BN	5	12	$2^{158}-2^{128}-2^{68}+1$	638	638	1	1275	7647
BLS12	4	12	$-2^{107}+2^{105}+2^{93}+2^{5}$	638	427	1.49	1276	7656
BLS24	4	24	$-2^{48}+2^{45}+2^{31}-2^{7}$	477	383	1.25	1914	11482

TABLE 7.2. Important parameters for the chosen KSS18, BN, BLS12, BLS24 curves. Here $\rho = \log p/\log r$.

Curve	Optimal ate pairing: $(P,Q)\mapsto$	$h(x)$
KSS18	$\left(f_{z,Q}\cdot f_{3,Q}^{p}\cdot \ell_{zQ,3pQ}(P)\right)^{(p^{18}-1)/r}$	$z+3x-x^4$
BN	$\left(f_{6z+2,Q}\cdot \ell_{(6z+2)Q,pQ}\cdot \ell_{(6z+2+p)Q,(-p^2)Q}(P)\right)^{(p^{12}-1)/r}$	$6z+2+x-x^2+x^3$
BLS12	$(f_{z,Q}(P))^{(p^{12}-1)/r}$	$z-x$
BLS24	$(f_{z,Q}(P))^{(p^{24}-1)/r}$	$z-x$

TABLE 7.3. Optimal ate pairings.

The pairing $w_{p^e,h}$ with $|h_i|\le r^{1/2}$ can be computed using two extended Miller functions of length approximately $\frac{1}{2}\log r$. We present a framework for constructing β Weil pairings (cf. Section 4.2.2), which can be computed using $2e$ extended Miller functions each of length approximately $(1/\varphi(k))\log r$. In particular, we prove that for a polynomial h for which $h(p)\equiv 0 \pmod r$, the following is a pairing:

$$\beta\colon G_1\times G_2\to G_T\;:\;(P,Q)\mapsto \prod_{i=0}^{e-1} w_{p,h}(p^iP,Q)^{p^{e-1-i}}. \tag{7.2}$$

To establish that (7.2) is a pairing, we require a few technical lemmas, building on the work of Hess and Vercauteren. Lemma 7.1 gives a pairing which is the product of Weil pairings consisting of Miller functions having ate-like lengths.

Lemma 7.1. For all positive integers s, the following map from $G_1\times G_2$ to G_T is a pairing:

$$(P,Q)\mapsto\left(\prod_{i=0}^{e-1}\left(\frac{f_{p^s,p^iQ}(P)}{f_{p^s,p^iP}(Q)}\right)^{p^{e-1-i}}\right)^{p^{k/2}-1}.$$

PROOF. Since the map

$$(P,Q)\mapsto\left(\frac{f_{p^e,Q}(P)}{f_{p^e,P}(Q)}\right)^{p^{k/2}-1}$$

is a pairing (Theorem 2.30), we apply Lemma 2.1(ii) to obtain

$$f_{p^e,P} = \prod_{i=0}^{e-1} (f_{p,p^iP})^{p^{e-1-i}}.$$

Hence, the result holds for $s = 1$.

Since

$$f_{p^s,P} = \prod_{j=0}^{s-1} (f_{p,p^jP})^{p^{s-1-j}},$$

we have

$$\begin{aligned}
\prod_{i=0}^{e-1} (f_{p^s,p^iP})^{p^{e-1-i}} &= \prod_{i=0}^{e-1} \left(\prod_{j=0}^{s-1} (f_{p,p^ip^jP})^{p^{s-1-j}} \right)^{p^{e-1-i}} \\
&= \prod_{j=0}^{s-1} \left(\prod_{i=0}^{e-1} (f_{p,p^i(p^jP)})^{p^{e-1-i}} \right)^{p^{s-1-j}} \\
&= \prod_{j=0}^{s-1} \left(f_{p^e,p^jP} \right)^{p^{s-1-j}}.
\end{aligned}$$

From this, we can observe that

$$\left(\prod_{i=0}^{e-1} \left(\frac{f_{p^s,p^iQ}(P)}{f_{p^s,p^iP}(Q)} \right)^{p^{e-1-i}} \right)^{p^{k/2}-1} = \left(\prod_{i=0}^{s-1} \left(\frac{f_{p^e,p^iQ}(P)}{f_{p^e,p^iP}(Q)} \right)^{p^{s-1-i}} \right)^{p^{k/2}-1}. \tag{7.3}$$

Since the map $(P, Q) \mapsto f_{p^e,Q}(P)$ is a pairing (Theorem 2.30), the right hand side of (7.3) is a product of pairings. □

The next lemma relates the previous pairing to the Weil pairing notation defined in (7.1).

Lemma 7.2. The following identity holds for all positive integers s:

$$\left(\prod_{i=0}^{e-1} \left(\frac{f_{p^s,p^iQ}(P)}{f_{p^s,p^iP}(Q)} \right)^{p^{e-1-i}} \right)^{p^{k/2}-1} = \prod_{i=0}^{e-1} w_{p^s}(p^iP, Q)^{p^{e-1-i}}.$$

Proof. Since the map $(P,Q) \mapsto f_{p^s,Q}(P)$ is a pairing, we have

$$\begin{aligned}\left(\prod_{i=0}^{e-1}\left(\frac{f_{p^s,p^iQ}(P)}{f_{p^s,p^iP}(Q)}\right)^{p^{e-1-i}}\right)^{p^{k/2}-1} &= \left(\prod_{i=0}^{e-1}\left(\frac{f_{p^s,Q}(p^iP)}{f_{p^s,p^iP}(Q)}\right)^{p^{e-1-i}}\right)^{p^{k/2}-1} \\ &= \prod_{i=0}^{e-1} w_{p^s}(p^iP,Q)^{p^{e-1-i}},\end{aligned}$$

as required. □

Finally, using the pairing relation from Lemma 7.2, we can obtain a pairing composed of Miller functions each with Vercauteren-style bound on the length.

Theorem 7.3. There exists h such that $|h_i| \le r^{1/\varphi(k)}$ and the following is a pairing:

$$\beta\colon G_1 \times G_2 \to G_T \;:\; (P,Q) \mapsto \prod_{i=0}^{e-1} w_{p,h}(p^iP,Q)^{p^{e-1-i}}.$$

Proof. Let $h(x) = \sum_{i=0}^{c} h_i x^i$ be given by Vercauteren's theorem (Theorem 3.7) and let $h(p) = rm$. Since

$$f_{r,P}^m = f_{p,h,P} \cdot \prod_{j=0}^{c} f_{p^j,P}^{h_j},$$

we have

$$w_r(P,Q)^m = w_{p,h}(P,Q) \cdot \prod_{j=0}^{c} w_{p^j}(P,Q)^{h_j}.$$

Hence

$$\begin{aligned}\prod_{i=0}^{e-1} w_{p,h}(p^iP,Q)^{p^{e-1-i}} &= \prod_{i=0}^{e-1}\left(w_r(p^iP,Q)^m \cdot \prod_{j=0}^{c} w_{p^j}(p^iP,Q)^{-h_j}\right)^{p^{e-1-i}} \\ &= \prod_{i=0}^{e-1} w_r(p^iP,Q)^{mp^{e-1-i}} \cdot \prod_{j=0}^{c}\left(\prod_{i=0}^{e-1} w_{p^j}(p^iP,Q)^{p^{e-1-i}}\right)^{-h_j},\end{aligned}$$

which by Lemmas 7.1 and 7.2 is a product of pairings. □

Using Theorem 7.3 and the polynomials h from Table 7.3, we found the β Weil pairings for BN, BLS12, KSS18 and BLS24 curves. For BN curves, we use our construction from Section 4.2.2. The β Weil pairings can be defined as follows:

$$\text{KSS18} : (P,Q) \mapsto \left[\left(\frac{f_{p,h,P}(Q)}{f_{p,h,Q}(P)}\right)^{p^2}\left(\frac{f_{p,h,pP}(Q)}{f_{p,h,Q}(pP)}\right)^{p}\frac{f_{p,h,p^2P}(Q)}{f_{p,h,Q}(p^2P)}\right]^{(p^9-1)(p^3+1)}, \tag{7.4}$$

$$\text{BN} : (P, Q) \mapsto \left[\left(\frac{f_{p,h,P}(Q)}{f_{p,h,Q}(P)} \right)^p \frac{f_{p,h,pP}(Q)}{f_{p,h,Q}(pP)} \right]^{(p^6-1)(p^2+1)}, \tag{7.5}$$

$$\text{BLS12} : (P, Q) \mapsto \left[\left(\frac{f_{z,P}(Q)}{f_{z,Q}(P)} \right)^p \frac{f_{z,pP}(Q)}{f_{z,Q}(pP)} \right]^{(p^6-1)(p^2+1)}, \tag{7.6}$$

$$\text{BLS24} : (P, Q) \mapsto \left[\frac{f_{z,P}^{p^3}(Q) \cdot f_{z,pP}^{p^2}(Q) \cdot f_{z,p^2P}^{p}(Q) \cdot f_{z,p^3P}(Q)}{f_{z,Q}^{p^3}(P) \cdot f_{z,Q}^{p^2}(pP) \cdot f_{z,Q}^{p}(p^2P) \cdot f_{z,Q}(p^3P)} \right]^{(p^{12}-1)(p^4+1)}. \tag{7.7}$$

For all four β Weil pairings, computing pP has approximately the same cost as computing zP.

7.1.2. Parallelization of pairings. Given two processors, the Weil pairing can be trivially parallelized since the numerator and denominator of the Weil pairing are independent operations. The ate pairing requires two serial operations, the Miller loop and the final exponentiation. The next lemma can be used to parallelize the computation of the Miller loop. We know of no way to parallelize the final exponentiation.

The method of Aranha et al. [3] for parallelizing the computation of a Miller function $f_{s,R}$ is the following. We first write $s = 2^w s_1 + s_0$ with $s_0 < 2^w$. Applying Lemmas 2.1 and 2.2, we obtain

$$f_{s,R} = f_{s_1,R}^{2^w} \cdot f_{2^w,s_1R} \cdot f_{s_0,R} \cdot \ell_{2^w s_1 R, s_0 R}/v_{sR}. \tag{7.8}$$

If s_0 is small, then the Miller function $f_{s_0,R}$ can be computed relatively cheaply. Thus the computation of $f_{s,R}$ can be parallelized by computing $f_{s_1,R}^{2^w}$ on one processor and f_{2^w,s_1R} on a second processor. The parameter w should be carefully selected in order to balance the time of the two function computations. The relevant criteria for selecting w include the Hamming weight of s_1 (which determines the number of additions in the Miller loop for the first function), and the cost of the w-fold squaring in the first function relative to the cost of computing s_1R in the second function. This idea can be extended to c processors by writing $s = 2^{w_{c-1}} s_{c-1} + \cdots + 2^{w_1} s_1 + s_0$.

Remark 7.1. (*unsuitability of composite-order BN curves*) Consider a BN curve at the 192-bit security level. For such a curve, we desire a (sparse) BN parameter z of approximately 160 bits. From the optimal pairing framework (Section 3.1.2), we choose a suitable vector $[2z, z+1, -z, z]$ corresponding to the following pairing (with the final exponentiation omitted):

$$(P, Q) \mapsto f_{2z,Q} \cdot f_{z+1,Q}^{p} \cdot f_{z,Q}^{-p^2} \cdot f_{z,Q}^{p^3} \cdot \ell_{(-zp^2)Q,(zp^3)Q} \cdot \ell_{p(z+1)Q,(-zp^2+zp^3)Q}(P).$$

Computation of the lines is relatively inexpensive. However, at first, it appears one must evaluate multiple Miller functions. Fortunately, for parameterized curves, one can (usually) rearrange terms such that the computational bottleneck is $f_{z,Q}$ with only a few lines comprising the remaining computation. In the above case, we obtain

$$(P,Q) \mapsto f_{z,Q}^{2+p-p^2+p^3} \cdot \ell_{zQ,zQ} \cdot \ell_{zpQ,pQ} \cdot \ell_{-zp^2Q,zp^3Q} \cdot \ell_{p(z+1)Q,(-zp^2+zp^3)Q}(P).$$

At the 192-bit security level, we require that r have a prime divisor of at least 384 bits. We can easily choose r to be (a 640-bit) prime. However, given that the optimal pairing framework gives a maximum Miller length of around $(\log n)/4$ for BN curves where n is a large prime divisor of r, we should be tempted to choose r with a 384-bit prime divisor. The fact that the coordinates of the vector $[2z, z+1, -z, z]$ have small coefficients when written in base z allowed us to write the pairings as a power of $f_{z,Q}$ multiplied by a few lines. However, for composite values of r, the vector with 96-bit elements which we obtain from the optimal pairing framework does not, in general, have coordinates which we can relate to each other. We would therefore require approximately 4 independent Miller functions, negating most of the benefit of computing an optimal pairing, rather than the Tate pairing. The possibility of choosing a vector whose elements are part of a short addition chain may still exist but the vectors produced by the LLL algorithm do not appear to maintain such structure. Thus, composite-order BN curves would appear to yield inferior performance compared to prime-order BN curves.

7.2. BLS12 pairings

In this section, we consider the BLS12 curve $Y^2 = X^3 + 4$ defined with the parameter selection

$$z = -2^{107} + 2^{105} + 2^{93} + 2^5$$

which yields a 638-bit prime p and a 427-bit prime r.

7.2.1. Extension field arithmetic for pairings with $\mathbf{k = 12}$. A tower extension for $\mathbb{F}_{p^{12}}$ can be constructed as follows:

$$\begin{aligned}
\mathbb{F}_{p^2} &= \mathbb{F}_p[u]/(u^2 - \beta), \text{ where } \beta \in \mathbb{F}_p,\\
\mathbb{F}_{p^6} &= \mathbb{F}_{p^2}[v]/(v^3 - \xi), \text{ where } \xi \in \mathbb{F}_{p^2}, \text{ and}\\
\mathbb{F}_{p^{12}} &= \mathbb{F}_{p^6}[w]/(w^2 - \gamma), \text{ where } \gamma \in \mathbb{F}_{p^6}.
\end{aligned}$$

For our choice of parameters, we have the optimal $\beta = -1$, $\xi = u+1$, $\gamma = v$. Table 7.4 gives the computational costs of the tower extension field arithmetic for curves with $k = 12$ in terms of a 640-bit multiplication (m_{640}) and inversion (i_{640}) in $\mathbb{F}_p$, with p a 638-bit prime.[1] The cost of additions is ignored because of their lower overall performance impact due to the larger field size in comparison with [2, 70]. Moreover, $\tilde{m}$, $\tilde{s}$, $\tilde{\imath}$ denote the cost of

[1]In the case of software implementation, this selection of the size of p facilitates the usage of *lazy reduction* techniques as recommended in [2, 70].

multiplication, squaring and inversion in $\mathbb{F}_{p^2}$ respectively.[2] $\mathbb{G}_{\Phi_6(p^2)}$ denotes the order-$\Phi_6(p^2)$ subgroup of $\mathbb{F}_{p^{12}}^*$, where Φ_k denotes the k-th cyclotomic polynomial.

Field	Mult.	Squaring	Inversion
$\mathbb{F}_{p^2}$	$\tilde{m} = 3m_{640}$	$\tilde{s} = 2m_{640}$	$\tilde{\imath} = 4m_{640} + i_{640}$
$\mathbb{F}_{p^6}$	$6\tilde{m}$	$\tilde{m} + 4\tilde{s}$	$9\tilde{m} + 3\tilde{s} + \tilde{\imath}$
$\mathbb{F}_{p^{12}}$	$18\tilde{m}$	$12\tilde{m}$	$23\tilde{m} + 11\tilde{s} + \tilde{\imath}$
$\mathbb{G}_{\Phi_6(p^2)}$	$18\tilde{m}$	$9\tilde{s}$	Conjugation

Operation	Cost
Sparse Mult.	$13\tilde{m}$
Sparser Mult.	$7\tilde{m}$
Compressed Squaring	$6\tilde{s}$
Simult. decompression of n field elements	$n(3\tilde{m} + 3\tilde{s})+$ $(n-1)3\tilde{m} + \tilde{\imath}$
$p/p^2/p^3$-Frobenius	$10m/15m/15m$

TABLE 7.4. Costs of arithmetic operations in a tower extension field $\mathbb{F}_{p^{12}}$.

7.2.2. Miller loop.

7.2.2. Miller loop. For the parameter selection $z = -2^{107} + 2^{105} + 2^{93} + 2^5$, the Miller loop computation of $f_{z,Q}$ requires 107 point doublings and associated line evaluations, 3 point additions with line evaluations, 109 sparse multiplications, and 106 squarings in $\mathbb{F}_{p^{12}}$. The computational costs of these operations can be found in [2, Table 1]. We obtain a BLS12 Miller loop cost of $107(3\tilde{m}+6\tilde{s}+4m_{640})+3(11\tilde{m}+2\tilde{s}+4m_{640})+109(13\tilde{m})+106(12\tilde{m})$ $= 3043\tilde{m} + 648\tilde{s} + 440m_{640} = 10865m_{640}$.

7.2.3. Final exponentiation. The final exponentiation consists of raising the Miller loop result $f \in \mathbb{F}_{p^k}$ to the $e = (p^k - 1)/r$-th power. This task can be broken into two parts since

$$e = (p^k - 1)/r = [(p^k - 1)/\Phi_k(p)] \cdot [\Phi_k(p)/r].$$

Computing $f^{(p^k-1)/\Phi_k(p)}$ is considered easy, costing only a few multiplications, inversions, and inexpensive p-th powerings in $\mathbb{F}_{p^k}$. Raising to the power $d = \Phi_k(p)/r$ is a more challenging task. Observing that p-th powering is much less expensive than multiplication, Scott et al. [84] give a systematic method for reducing the expense of exponentiating by d. In the case of BLS12 curves, it can be shown that the exponent d can be written as

[2]For further details on how these costs were deduced, the reader is referred to [2, 70].

$d = \lambda_0 + \lambda_1 p + \lambda_2 p^2 + \lambda_3 p^3$ where

$$\begin{aligned}\lambda_0(z) &= z^5 - 2z^4 + 2z^2 - z + 3,\\ \lambda_1(z) &= z^4 - 2z^3 + 2z - 1,\\ \lambda_2(z) &= z^3 - 2z^2 + z,\\ \lambda_3(z) &= z^2 - 2z + 1.\end{aligned}$$

The exponentiation f^d can be computed using the following addition-subtraction chain:

$$\begin{aligned}f \to f^{-2} \to f^z \to f^{2z} \to f^{z-2} \to f^{z^2-2z} \to f^{z^3-2z^2} \to f^{z^4-2z^3}\\ \to f^{z^4-2z^3+2z} \to f^{z^5-2z^4+2z^2},\end{aligned}$$

which requires 5 exponentiations by z, 2 multiplications in $\mathbb{F}_{p^{12}}$, and 2 cyclotomic squarings. This allows f^d to be computed as

$$f^d = f^{z^5-2z^4+2z^2} \cdot (f^{z-2})^{-1} \cdot f \cdot (f^{z^4-2z^3+2z} \cdot f^{-1})^p \cdot (f^{z^3-2z^2} \cdot f^z)^{p^2} \cdot (f^{z^2-2z} \cdot f)^{p^3},$$

which requires an additional 8 multiplications in $\mathbb{F}_{p^{12}}$ and 3 Frobenius maps. This implies that the hard part of the final exponentiation requires 2 cyclotomic squarings, 5 exponentiations by z, 10 multiplications in $\mathbb{F}_{p^{12}}$, and 3 Frobenius maps.

In total, the cost of computing the final exponentiation is 1 inversion in $\mathbb{F}_{p^{12}}$, 2 cyclotomic squarings, 12 multiplications in $\mathbb{F}_{p^{12}}$, 4 Frobenius maps, and 5 exponentiations by z. It can be shown that exponentiation by our choice of the z parameter requires 107 compressed squarings, simultaneous decompression of 4 field elements, and 3 multiplications in $\mathbb{F}_{p^{12}}$ when Karabina's exponentiation technique [48] is employed. The cost of an exponentiation by z is $107(6\tilde{s}) + 4(3\tilde{m} + 3\tilde{s}) + 3(3\tilde{m}) + \tilde{\imath} + 3(18\tilde{m}) = 75\tilde{m} + 654\tilde{s} + \tilde{\imath}$, whence the total cost of the final exponentiation is $(23\tilde{m} + 11\tilde{s} + \tilde{\imath}) + 2(9\tilde{s}) + 12(18\tilde{m}) + 60m_{640} + 5(75\tilde{m} + 654\tilde{s} + \tilde{\imath}) = 614\tilde{m} + 3299\tilde{s} + 6\tilde{\imath} = 8464m_{640} + 6i_{640}$.

7.2.4. Optimal pairing cost. From the above, we conclude that the estimated cost of the optimal ate pairing for our chosen BLS12 curve is

$$10865m_{640} + 8464m_{640} + 6i_{640} = 19329m_{640} + 6i_{640}.$$

7.2.5. Parallelization. Figure 7.1 illustrates the execution path for the β Weil pairing (7.6) when the four Miller functions are computed in parallel using 4 processors. As with the optimal ate pairing, Lemmas 2.1 and 2.2 were repeatedly applied to each Miller function in the β Weil pairing in order to obtain a parallel implementation using 8 processors.

7.3. KSS18 pairings

In this section, we consider the KSS18 curve $Y^2 = X^3 + 2$ defined with the parameter selection

$$z = -2^{64} - 2^{61} + 2^{46} + 2^{12}.$$

$$\begin{tikzcd}
1. \; f_{z,Q}(P) \arrow[rr] & & f_{z,Q}^{p}(P) \arrow[r] & f_{z,Q}^{p}(P)\cdot f_{z,Q}(pP) \\
2. \; pP \arrow[r] & f_{z,Q}(pP) \arrow[ur] & & \\
3. \; f_{z,P}(Q) \arrow[rr] & & f_{z,P}^{p}(Q) \arrow[r] & f_{z,P}^{p}(Q)\cdot f_{z,pP}(Q) \\
4. \; pP \arrow[r] & f_{z,pP}(Q) \arrow[ur] & &
\end{tikzcd}$$

FIGURE 7.1. Execution path for computing the β Weil pairing for BLS12 curves on 4 processors.

7.3.1. Extension field arithmetic for pairings with $\mathbf{k = 18}$. An element in $\mathbb{F}_{p^{18}}$ can be represented using the following towering scheme:

$$\begin{aligned}
\mathbb{F}_{p^3} &= \mathbb{F}_p[u]/(u^3+2),\\
\mathbb{F}_{p^6} &= \mathbb{F}_{p^3}[v]/(v^2-u),\\
\mathbb{F}_{p^{18}} &= \mathbb{F}_{p^6}[w]/(w^3-v).
\end{aligned}$$

Table 7.5 gives the computational costs of the tower extensions field arithmetic for curves with $k = 18$, where m_{512}, i_{512} denote the cost of multiplication and inversion in $\mathbb{F}_p$, with p a 512-bit prime. Moreover, $\hat{m}$, $\hat{s}$, $\hat{\imath}$ denote the cost of multiplication, squaring and inversion in $\mathbb{F}_{p^3}$ respectively.

Field	Mult.	Squaring	Inversion
$\mathbb{F}_{p^3}$	$\hat{m} = 6m_{512}$	$\hat{s} = 5m_{512}$	$\hat{\imath} = 12m_{512} + i_{512}$
$\mathbb{F}_{p^6}$	$3\hat{m}$	$2\hat{m}$	$2\hat{m} + 2\hat{s} + \hat{\imath}$
$\mathbb{F}_{p^{18}}$	$18\hat{m}$	$11\hat{m}$	$20\hat{m} + 8\hat{s} + \hat{\imath}$
$\mathbb{G}_{\Phi_6(p^3)}$	$18\hat{m}$	$6\hat{m}$	Conjugation

Operation	Cost
Sparse Mult.	$13\hat{m}$
Sparser Mult.	$7\hat{m}$
Compressed Squaring	$4\hat{m}$
Simult. decompression of n field elements	$n(3\hat{m} + 3\hat{s}) + (n-1)3\hat{m} + \hat{\imath}$
pth-Frobenius	$15m$

TABLE 7.5. Costs of arithmetic operations in a tower extension field $\mathbb{F}_{p^{18}}$.

7.3.2. Computation of the optimal ate pairing. For the parameter selection $z = -2^{64} - 2^{51} + 2^{46} + 2^{12}$, the Miller loop executes 64 point doublings with line evaluations, 4 point additions with line evaluations, 67 sparse multiplications and 63 squarings in $\mathbb{F}_{p^{18}}$. We obtain a KSS18 Miller loop cost of $64(3\hat{m}+6\hat{s}+6m_{512})+4(11\hat{m}+2\hat{s}+6m_{512})+67(13\hat{m})+63(11\hat{m}) = 1800\hat{m} + 392\hat{s} + 408m_{512} = 13168m_{512}$. Furthermore, the final step executes

1 squaring in $\mathbb{F}_{p^{18}}$, one p-power Frobenius, 1 multiplication in $\mathbb{F}_{p^{18}}$, 2 point additions with line evaluation, one point doubling with line evaluation, 1 sparse multiplication, 1 sparser multiplication, and the computation of the isomorphism $\psi(Q)$. Thus the KSS18 final step cost is $11\hat{m} + 18\hat{m} + 2(11\hat{m} + 2\hat{s} + 6m_{512}) + 3\hat{m} + 6\hat{s} + 6m_{512} + 20\hat{m} + 28m_{512} = 74\hat{m} + 10\hat{s} + 40m_{512} = 534m_{512}$. The final exponentiation executes in total one inversion in $\mathbb{F}_{p^{18}}$, 8 cyclotomic squarings, 54 multiplications in $\mathbb{F}_{p^{18}}$, 29 p-power Frobenius, and 7 exponentiations by z (Section 5.2.4). The computational cost of an exponentiation by z is 64 compressed squarings, decompression of 4 field elements and 3 multiplications in $\mathbb{F}_{p^{18}}$, for a total cost of $64(6\hat{s}) + 4(3\hat{s} + 3\hat{m}) + 9\hat{m} + \hat{\imath} + 3(18\hat{m}) = 75\hat{m} + 396\hat{s} + \hat{\imath}$. Hence, the total cost of the final exponentiation is $20\hat{m} + 8\hat{s} + \hat{\imath} + 8(6\hat{m}) + 54(18\hat{m}) + 435m_{512} + 7(75\hat{m} + 396\hat{s} + \hat{\imath}) = 1565\hat{m} + 2780\hat{s} + 8\hat{\imath} + 435m_{512} = 23821m_{512} + 8i_{512}$ Finally, the total cost of computing the KSS18 optimal ate pairing is

$$13168m_{512} + 534m_{512} + 23821m_{512} + 8i_{512} = 37523m_{512} + 8i_{512}.$$

7.3.3. Computation of the β Weil pairing. The most expensive part of the β Weil pairing for KSS18 curves (7.4) are the six Miller functions $f_{z,R}$. For parallel implementation using 4 cores, repeated applications of Lemmas 2.1 and 2.2 can be used to write $z = 2^w z_1 + z_0$ such that $f_{z,R}$ can be computed in the following way:

$$f_{z,R} = f_{z_1,R}^{2^w} \cdot f_{2^w, z_1 R} \cdot f_{z_0,R} \cdot (\ell_{2^w} \cdot z_1 R, z_0 R)/v_{zR}.$$

For the KSS18 parameter $z = -2^{64} - 2^{51} + 2^{46} + 2^{12}$, we chose $w = 36$, $z_1 = -2^{28} + 2^{15} + 2^{10}$, $z_0 = 2^{12}$ and split the two most expensive Miller functions $f_{z,Q}^p(pP)$ and $f_{z,Q}(p^2P)$. Figure 7.2 illustrates an execution path. At the end, it is necessary for each core to compute the additional functions $(f_{3,R}^p \cdot \ell_{zR,3pR})^{p^i}$ and the exponentiation by $(p^9 - 1) \cdot (p^3 + 1)$.

$$\begin{tikzcd}
1.\ pP \arrow[rr] & & \frac{1}{f_{z,Q}^{p^2}(P) \cdot f_{z_1,Q}^{2^w p}(p(P)} \arrow[r] & \frac{f_{z,P}^{p^2}(Q)}{f_{z,Q}^{p^2}(P) \cdot f_{z,Q}^{p}(pP)} \\
2.\ pP \arrow[rr] & & \frac{f_{z,P}^{p^2}(Q)}{f_{2^w,z_1Q}^{p}(pP) \cdot f_{z_0,Q}^{p}(pP)} \arrow[ur] & \\
3.\ pP \arrow[r] & p^2P \arrow[r] & \frac{f_{z,pP}^{p}(Q)}{f_{z_1,Q}^{2^w}(p^2P)} \arrow[r] & \frac{f_{z,pP}^{p}(Q) \cdot f_{z,p^2P}(Q)}{f_{z,Q}(p^2P)} \\
4.\ pP \arrow[r] & p^2P \arrow[r] & \frac{f_{z,p^2P}(Q)}{f_{2^w,z_1(Q)}(p^2P) \cdot f_{z_0,Q}(p^2P)} \arrow[ur] &
\end{tikzcd}$$

FIGURE 7.2. Execution path for computing the β Weil pairing for KSS18 curves on 4 processors.

For the case of an 8-core implementation, we simply reschedule these functions so that each core takes approximately the same time.

7.4. BN pairings

In this section, we consider the BN curve $Y^2 = X^3 + 5$ defined with the parameter selection

$$z = 2^{158} - 2^{128} - 2^{68} + 1.$$

The extension fields are

$$\begin{aligned}\mathbb{F}_{p^2} &= \mathbb{F}_p[u]/(u^2+1),\\ \mathbb{F}_{p^6} &= \mathbb{F}_{p^2}[v]/(v^3-\xi), \text{with } \xi = u+2\\ \mathbb{F}_{p^{12}} &= \mathbb{F}_{p^6}[w]/(w^2-v).\end{aligned}$$

7.4.1. Computation of the optimal ate pairing. The Miller loop executes 160 point doublings with line evaluations, 6 point additions with line evaluations, 164 sparse multiplications, 1 sparser multiplication and 159 squarings in $\mathbb{F}_{p^{12}}$. We obtain a BN Miller loop cost of $160(3\tilde{m} + 6\tilde{s} + 4m_{640}) + 6(11\tilde{m} + 2\tilde{s} + 4m_{640}) + 164(13\tilde{m}) + 7\tilde{m} + 159(12\tilde{m}) = 4593\tilde{m} + 972\tilde{s} + 664m_{640} = 16387m_{640}$.

Furthermore, the final step executes $\psi(Q)$, $\psi^2(Q)$, 2 point additions with line evaluation, 1 sparser multiplication and 1 multiplication in $\mathbb{F}_{p^{12}}$. The p-th power Frobenius can be computed at a cost of about $5m_{640}$ and the p^2-th power Frobenius can be computed at a cost of about $4m_{640}$. Thus the BN final step cost is $2(11\tilde{m} + 2\tilde{s} + 4m_{640}) + 7\tilde{m} + 18\tilde{m} + 9m_{640} = 47\tilde{m} + 4\tilde{s} + 17m_{640} = 166m_{640}$. The final exponentiation executes in total 1 inversion in $\mathbb{F}_{p^{12}}$, 3 cyclotomic squarings, 12 multiplications in $\mathbb{F}_{p^{12}}$, 2 p-th power Frobenius, 1 p^2-th power Frobenius, 1 p^3-th power Frobenius, and 3 exponentiations by z (Section 5.2.1). The computational cost of an exponentiation by z is: 158 compressed squarings, decompression of 3 field elements and 3 multiplications in $\mathbb{F}_{p^{12}}$, for a total cost of $158(6\tilde{s}) + 3(3\tilde{s} + 3\tilde{m}) + 6\tilde{m} + \tilde{\imath} + 3(18\tilde{m}) = 69\tilde{m} + 957\tilde{s} + \tilde{\imath}$. Hence, the total cost of the final exponentiation is $23\tilde{m} + 11\tilde{s} + \tilde{\imath} + 3(9\tilde{s}) + 12(18\tilde{m}) + 50m_{640} + 3(69\tilde{m} + 957\tilde{s} + \tilde{\imath}) = 446\tilde{m} + 2909\tilde{s} + 62m_{640} + 4i_{640} = 7218m_{640} + 4i_{640}$. Finally, the total cost of computing the BN optimal ate pairing is

$$16387m_{640} + 166m_{640} + 7218m_{640} + 4i_{640} = 23771m_{640} + 4i_{640}.$$

7.4.2. Computation of the β Weil pairing. For BN curves, we consider the β pairing from Section 4.2.2. Lemmas 2.1 and 2.2 were repeatedly applied in order to estimate the cost of a parallel implementation using 8 processors.

7.5. BLS24 pairings

In this section, we consider the BLS24 curve $Y^2 = X^3 + 1$ defined with the parameter selection

$$z = -2^{48} + 2^{45} + 2^{31} - 2^7.$$

7.5.1. Extension field arithmetic for pairings with $\mathbf{k = 24}$. An element in $\mathbb{F}_{p^{24}}$ can be represented using the following towering scheme:

$$\begin{aligned} \mathbb{F}_{p^2} &= \mathbb{F}_p[i]/(i^2+1), \\ \mathbb{F}_{p^4} &= \mathbb{F}_{p^2}[u]/(u^2-\xi), \text{ with } \xi = i+1, \\ \mathbb{F}_{p^{12}} &= \mathbb{F}_{p^4}[v]/(v^2-u), \\ \mathbb{F}_{p^{24}} &= \mathbb{F}_{p^{12}}[w]/(w^2-v). \end{aligned}$$

Table 7.6 gives the computational costs of the tower extension field arithmetic for curves with $k = 24$, where m_{480} and i_{480} denote the cost of multiplication and inversion in $\mathbb{F}_p$, with p a 479-bit prime. Moreover, $\tilde{m}$, $\tilde{s}$, $\tilde{\imath}$ denote the cost of multiplication, squaring and inversion in $\mathbb{F}_{p^2}$ respectively.

Field	Mult.	Squaring	Inversion
$\mathbb{F}_{p^2}$	$\tilde{m} = 3m_{480}$	$\tilde{s} = 2m_{480}$	$\tilde{\imath} = 4m_{480} + i_{480}$
$\mathbb{F}_{p^4}$	$3\tilde{m}$	$2\tilde{m}$	$2\tilde{m} + 2\tilde{s} + \tilde{\imath}$
$\mathbb{F}_{p^{12}}$	$18\tilde{m}$	$12\tilde{m}$	$23\tilde{m} + 11\tilde{s} + \tilde{\imath}$
$\mathbb{F}_{p^{24}}$	$54\tilde{m}$	$36\tilde{m}$	$83\tilde{m} + 11\tilde{s} + \tilde{\imath}$
$\mathbb{G}_{\Phi_6(p^4)}$	$54\tilde{m}$	$18\tilde{m}$	Conjugation

	Operation Count
Sparse Mult.	$39\tilde{m}$
Sparser Mult.	$21\tilde{m}$
Compressed Squaring	$12\tilde{m}$
Simult. decompression of n field elements	$(2n-1)(9\tilde{m}) + n(6\tilde{m})$ $+2\tilde{m} + 2\tilde{s} + \tilde{\imath}$
$p/p^2/p^3$-Frobenius	$45m$

TABLE 7.6. Costs of arithmetic operations in a tower extension field $\mathbb{F}_{p^{24}}$.

7.5.2. Computation of the optimal ate pairing. The Miller loop executes 48 point doublings with line evaluations, 4 point additions with line evaluations, 51 sparse multiplications and 47 squarings in $\mathbb{F}_{p^{24}}$. We obtain a BLS24 Miller loop cost of $48(21\tilde{m} + 8m_{480}) + 4(37\tilde{m} + 8m_{480}) + 51(39\tilde{m}) + 47(36\tilde{m}) = 4837\tilde{m} + 416m_{480} = 14927m_{480}$. The computation of the final exponentiation requires 1 inversion, 9 exponentiations by z, 14 multiplications in $\mathbb{F}_{p^{24}}$, 2 cyclotomic squarings, and 8 p-th power Frobenius operations. Moreover, the cost of an exponentiation by z is 48 compressed squarings, decompression of 4 field elements and 3 multiplications in $\mathbb{F}_{p^{24}}$, for a total cost of $48(12\tilde{m})+87\tilde{m}+\tilde{\imath}+3(54\tilde{m}) = 827\tilde{m}+2\tilde{s}+\tilde{\imath}$. Hence, the total cost of the final exponentiation is $(83\tilde{m}+11\tilde{s}+\tilde{\imath})+9(827\tilde{m}+2\tilde{s}+\tilde{\imath})+14(54\tilde{m})+2(18\tilde{m})+360m_{480} = 8318\tilde{m}+13\tilde{s}+400m_{480}+10i_{480} = 25380m_{480}+10i_{480}$.

Finally, the total cost of computing the BLS24 optimal ate pairing is

$$14927m_{480} + 25380m_{480} + 10i_{480} = 40307m_{480} + 10i_{480}.$$

7.5.3. Computation of the β Weil pairing. Since $4 \mid e$ where $e = k/d$, the parallelization procedure for the β Weil pairing (7.7) on 2, 4 and 8 cores is straightforward: with 2 cores, each core computes 4 Miller functions; with 4 cores, each core computes 2 Miller functions; and with 8 cores: each core computes 1 Miller function.

7.6. Comparisons

7.6.1. Estimates for serial implementations of the optimal ate pairings. The customary way to estimate the cost of a pairing is to count multiplications in the underlying finite fields. Notice that in the case of software implementations in modern desktop platforms, field elements $a \in \mathbb{F}_p$ can be represented with $\ell = 1 + \lfloor \log_2(p) \rfloor$ binary coefficients a_i packed in $n_{64} = \lceil \frac{\ell}{64} \rceil$ 64-bit processor words. If Montgomery representation is used to implement field multiplication in $\mathbb{F}_{p_{640}}$ and $\mathbb{F}_{p_{512}}$ with complexity $O(2n_{64}^2 + n_{64})$, then it is reasonable to estimate that we have $m_{640} \approx (210/136) \cdot m_{512} \approx 1.544 \cdot m_{512}$.

Table 7.7 summarizes the costs in terms of finite field multiplications for computing the optimal ate pairing over our choice of KSS18, BN, BLS12 and BLS24 curves at the 192-bit security level.[3] As can be seen, our estimates predict that the optimal ate pairing over BLS12 curves is the most efficient choice at the 192-bit security level, with KSS18, BN and BLS24 curves being significantly slower. The main computational bottleneck for BLS24 curves is their very expensive final exponentiation.

7.6.2. Estimates for multi-core implementations of the optimal ate and β Weil pairings. Table 7.8 (see also Figure 7.3) shows estimated speedups for the parallel version of the optimal ate pairing using the partitions in Table 7.9 and all the β Weil pairing variants considered here. All speedup factors are with respect to the serial version of the KSS18 optimal ate pairing. It can be seen that the estimated performance for BLS12 curves when using 8 cores is of a factor-3.29 acceleration, which is the highest speedup we obtain. Perhaps the most notable observation from Table 7.8 is that, for eight-core implementations, the β Weil pairing becomes more efficient than the optimal ate pairing for all the four curves considered.

[3]In the case of BN and KSS18 curves it is necessary to compute several extra lines and Frobenius maps. We refer to these steps as the "Final step". We stress that there is no analogous final step in the case of BLS12 and BLS24 curves.

Curve	Phase	Mult. in $\mathbb{F}_p$	Mult. in $\mathbb{F}_{p_{512}}$
KSS18	Miller Loop	$13168m_{512}$	$13168m_{512}$
	Final Step	$534m_{512}$	$534m_{512}$
	Final Exp.	$23821m_{512}$	$23821m_{512}$
	ML + FS + FE	$37523m_{512}$	$37523m_{512}$
BN	Miller Loop	$16387m_{640}$	$25301m_{512}$
	Final Step	$166m_{640}$	$256m_{512}$
	Final Exp.	$7218m_{640}$	$11145m_{512}$
	ML + FS + FE	$23771m_{640}$	$36702m_{512}$
BLS12	Miller Loop	$10865m_{640}$	$16775m_{512}$
	Final Exp.	$8464m_{640}$	$13068m_{512}$
	ML + FE	$19329m_{640}$	$29843m_{512}$
BLS24	Miller Loop	$14927m_{480}$	$14927m_{512}$
	Final Exp.	$25412m_{480}$	$25412m_{512}$
	ML + FE	$40339m_{480}$	$40339m_{512}$

TABLE 7.7. Cost estimates of the optimal ate pairing for KSS18, BN, BLS12 and BLS24 curves at the 192-bit security level. Note that $m_{480} = m_{512}$ in a 64-bit processor.

	Number of threads			
Estimated speedup KSS18	**1**	**2**	**4**	**8**
Optimal ate	1.00	1.17	1.28	1.33
β Weil	0.47	0.91	1.54	2.51
Estimated speedup BN	**1**	**2**	**4**	**8**
Optimal ate	1.02	1.36	1.61	1.76
β Weil	0.41	0.81	1.42	2.16
Estimated speedup BLS12	**1**	**2**	**4**	**8**
Optimal ate	1.26	1.56	1.76	1.88
β Weil	0.64	1.25	2.20	3.29
Estimated speedup BLS24	**1**	**2**	**4**	**8**
Optimal ate	0.93	1.05	1.12	1.14
β Weil	0.40	0.78	1.49	2.39

TABLE 7.8. Estimated speedups for the parallel version of the optimal ate pairing versus the β Weil pairing. All speedup factors are with respect to the serial version of the KSS18 optimal ate pairing.

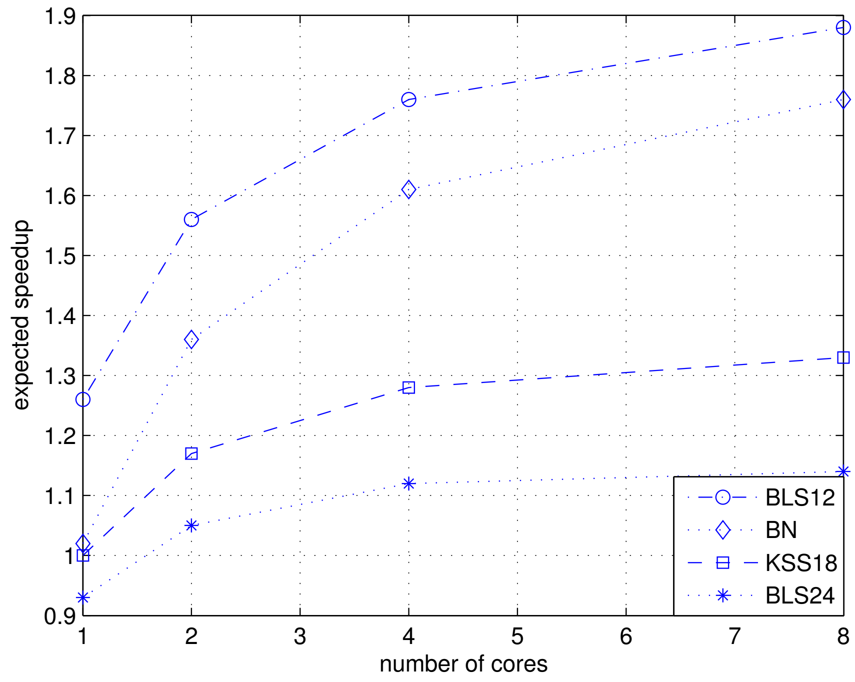

FIGURE 7.3. Expected speedups for KSS18, BN, BLS12 and BLS24 optimal ate pairings at the 192-bit security level. All speedup factors are with respect to the serial version of the KSS18 optimal ate pairing.

	Number of threads (c)		
Curve	2	4	8
KSS18	36	54, 39, 21	63, 58, 52, 45, 36, 26, 14
BN	86	129, 93, 50	149, 137, 122, 105, 85, 61, 33
BLS12	57	85, 61, 33	98, 90, 81, 70, 56, 40, 21
BLS24	26	38, 28, 15	44, 41, 37, 32, 26, 19, 10

TABLE 7.9. Parameters w_i, $0 < i < c$, which define the partition of the form $s = 2^{w_{c-1}}s_{c-1} + \cdots + 2^{w_1}s_1 + s_0$ for splitting the Miller loop according to Equation (7.8) when computing a multi-thread optimal ate pairing among c processing units.

7.6.3. Timings. We implemented the KSS18, BN, BLS12 and BLS24 pairings following the techniques described in [2] on two different 64-bit 32nm platforms, an Intel Core i5 540M Nehalem and an Intel Core i7 2630QM Sandy Bridge. Field arithmetic was implemented in Assembly for maximum efficiency and high-level code was implemented in the C programming language. The GCC 4.7.0 compiler suite was used with compilation flags for loop unrolling, inlining of small functions to reduce function call overheads, and optimization level `-O3`. The implementation was done on top of the RELIC cryptographic toolkit [1].

The $m_{640} \approx 1.544 \cdot m_{512}$ estimate used above was experimentally confirmed with carefully crafted Assembly code for multiplication and Montgomery reduction. Implementing the double-precision arithmetic needed for efficient application of lazy reduction proved to be slightly cumbersome due to the exhaustion of the 16 general-purpose registers available in the target platform (one of the registers is mostly reserved for keeping track of stack memory, aggravating the effect). Naturally, this issue had a bigger performance impact on the larger 638-bit field, introducing higher penalties for reading and writing values stored into memory. By using a very efficient implementation of the Extended Euclidean Algorithm imported from the GMP[4] library, we obtained inversion-to-multiplication ratios in $\mathbb{F}_p$ of around 16, suggesting the use of the projective coordinate system instead of the affine coordinates recommended in [80] and [51], even after considering the action of the norm map to simplify the inversion operation in extension fields. Affine coordinates were only competitive for the BLS24 curve.

The resulting timings for the two platforms are presented in Table 7.10 (measured with the Turbo Boost feature disabled). Timings for the parallel implementation of pairings which were estimated to be slower than the reference performance of the KSS18 pairing are omitted. We obtained results confirming our performance estimates, i.e., the BLS12 curve is the most efficient choice for pairing computation at the 192-bit security level across all the considered scenarios. In particular, our fastest serial implementation on the Intel Core i5 Nehalem machine can compute a pairing in approximately 19 million cycles, more than 3 times faster than the current state-of-the-art. The previous speed record for a single pairing computation without precomputation at this security level was presented in [80, Table 2, column 4 halved] and achieves a latency of 60 million cycles on a very similar machine when a factor of 1.22 is applied to the timings to adjust for the effect of Turbo Boost.[5] Additionally, the β Weil pairing presents itself as the most efficient and scalable choice of pairing in a multiprocessor machine with more than 4 processing units.

[4]GNU Multiple Precision Arithmetic Library: http://www.gmplib.org

[5]This was confirmed with the author via private communication.

	Number of threads			
Platform 1 – Intel Core i5 Nehalem 32nm	**1**	**2**	**4***	**8***
KSS18 optimal ate – latency	23.40	20.91	19.75	19.17
KSS18 optimal ate – speedup	1.00	1.12	1.18	1.22
KSS18 β Weil – latency	–	–	15.04	9.18
KSS18 β Weil – speedup	–	–	1.56	2.55
BN optimal ate – latency	23.22	17.28	14.63	13.40
BN optimal ate – speedup	1.01	1.35	1.59	1.73
BN β Weil – latency	–	–	16.65	11.17
BN β Weil – speedup	–	–	1.39	2.08
BLS12 optimal ate – latency	18.67	15.15	13.49	12.58
BLS12 optimal ate – speedup	1.25	1.54	1.73	1.86
BLS12 β Weil – latency	–	19.38	10.80	7.24
BLS12 β Weil – speedup	–	1.21	2.17	3.23
BLS24 optimal ate – latency	26.32	24.00	22.82	22.27
BLS24 optimal ate – speedup	0.89	0.98	1.03	1.05
BLS24 β Weil – latency	–	–	17.83	10.26
BLS24 β Weil – speedup	–	–	1.31	2.28
Platform 2 – Intel Core i7 Sandy Bridge 32nm	**1**	**2**	**4**	**8***
KSS18 optimal ate – latency	17.73	15.76	14.95	14.52
KSS18 optimal ate – speedup	1.00	1.12	1.19	1.22
KSS18 β Weil – latency	–	–	11.36	6.97
KSS18 β Weil – speedup	–	–	1.56	2.54
BN optimal ate – latency	17.43	13.00	10.98	10.05
BN optimal ate – speedup	1.02	1.36	1.61	1.76
BN β Weil – latency	–	–	12.58	8.45
BN β Weil – speedup	–	–	1.41	2.10
BLS12 optimal ate – latency	14.08	11.41	10.11	9.48
BLS12 optimal ate – speedup	1.26	1.55	1.75	1.87
BLS12 β Weil – latency	–	14.58	8.13	5.47
BLS12 β Weil – speedup	–	1.22	2.18	3.24
BLS24 optimal ate – latency	19.97	18.27	17.21	16.86
BLS24 optimal ate – speedup	0.89	0.97	1.03	1.05
BLS24 β Weil – latency	–	–	13.75	7.90
BLS24 β Weil – speedup	–	–	1.29	2.24

TABLE 7.10. Experimental results for serial/parallel executions of the KSS18, BN and BLS12 optimal ate and β Weil pairings. Timings are presented in millions of clock cycles. The speedups are with respect to the serial version of the KSS18 optimal ate pairing. The columns marked with (*) present estimates based on per-thread data.

CHAPTER 8

A Privacy-Preserving Ecash-based Road-Tolling Protocol

This chapter describes a security application where BLS signatures play an important role due to their efficient signature generation and small signatures. Electronic Toll Pricing (ETP) aims to improve road tolling by collecting toll fares electronically and without the need to slow down vehicles. In most ETP schemes, drivers are charged periodically based on the locations, times, distances or durations travelled. ETP is sometimes used to reduce traffic congestion as vehicles tend to avoid peak drive times due to the extra fees associated with them, such as with the London congestion charge [56]. Usually, these schemes require the use of an *on-board unit* in the vehicle. As opposed to traditional stop-and-pay tollbooths, ETP schemes can be used in a more pervasive manner, potentially making, for example, all roads in a downtown core into toll roads.

Many ETP schemes are currently deployed, such as e-TAG in Australia [45] and E-ZPass in the United States [73]. Singapore was the first to implement an ETP system to reduce congestion in 1998 [21]. Although these systems are efficient, they require a great deal of knowledge regarding driving habits in order to operate correctly. In particular, these systems store time, location, and identity, making it easy to build a profile of a driver's daily travel. In the United States, E-ZPass records can be obtained by court order in several jurisdictions for use in civil matters such as divorce [66]. Given that a high level of privacy can be obtained using traditional stop-and-pay toll booths, an ETP protocol that maintains a comparable level of privacy would be ideal.

We propose two ETP schemes in which tokens, based on BLS or RSA-FDH signatures, are broadcast periodically from the driver's vehicle. If a driver is caught not broadcasting a valid token, the driver is appropriately penalized. In this manner, we ensure that drivers behave honestly at all times, so long as they cannot discern when someone is monitoring their broadcasts. In our first scheme, we offer privacy guarantees similar to those offered by other privacy-preserving schemes. In particular, the central authority only learns the location data corresponding to times and places where the vehicle is physically observed. In our second scheme, we examine the cost of a system that does not require the collection of *any* private data of honest drivers. Both of these schemes rely on simple primitives such as RSA Full Domain Hash [26] and Chaum's ecash [23]. This is in contrast to the comparatively expensive zero-knowledge proofs and secure two-party computations needed by other ETP protocols, such as VPriv [72], PrETP [4], and Milo [58].

We present SPEcTRe, a suite of protocols for electronic toll pricing. The SPEcTRe spot-record scheme records time-location information where drivers are physically observed. This is comparable to other schemes, but runs much faster, requiring only a modest amount of computational power to support one million vehicles. All the state-of-art ETP protocols including our spot-record scheme keep records of a small amount of location-time tuples in order to combat dishonest drivers. However, the SPEcTRe no-record scheme stores *no private information* of drivers, while still being capable of detecting cheaters.

8.1. Related work

Recently, much work has been done to build privacy-preserving schemes for electronic toll pricing and other related driving problems. For instance, work has been done to extend the problem to insurance pricing and speeding detection. Popa et al.'s VPriv [72] provides a practical protocol for computing path functions for several driving-related problems while maintaining a high level of privacy. The protocol uses a secure two-party computation to make sure drivers cannot cheat on the total tolling price if they do not cheat on the tokens they have uploaded. To ensure users upload the correct data, the authority is required to randomly record some {license plate, location, time} tuples, and then challenge drivers with these records during payment. It is important that the number of random observations, or "spot-checks", are kept to a moderate amount to maintain drivers' privacy. A measurement is performed to show that a modest hardware infrastructure can easily serve one million vehicles. VPriv is one of several recent papers that attempt to solve the problem of location privacy through the use of zero-knowledge proofs and secure multi-party computation.

Published a year after VPriv, Balasch et al. propose the PrETP scheme [4] with similar goals. They define a cryptographic protocol, Optimistic Payment, and prove it secure using zero-knowledge proofs and the RSA assumption. Unlike VPriv, they do not use secure two-party computations. Instead, the Toll Charger requires homomorphic commitments from clients, and the Toll Service Provider asks clients to open commitments to certain location-time tuples corresponding to its random spot-checks. For each client, only the total payment amount and location-time tuples in the physical vicinity of random spot-checks are revealed. They construct an on-board unit and analyze their system from a security, legal, performance, and cost perspective. Based on PrETP, Meiklejohn et al.'s Milo [58] furthermore considers the possibility that drivers may collude to learn the spot-checking locations. By utilizing blind identity-based encryption, these locations are not revealed to drivers.

The use of ecash to pay for driving has been suggested by Chaum and others [6, 24, 57] but the concept has not been used in a pervasive electronic tolling scheme until now. In particular, it is merely suggested that ecash replace physical currency. However, we have found that we can improve on this naive implementation by reducing the number of tolling points while maintaining driver honesty and privacy.

In VPriv, PrETP and Milo, spot-checks are necessary to uncover cheating drivers. SPEcTRe maintains the same level of privacy as these other schemes for drivers, in the sense that in all these schemes, location-time information of a vehicle is only revealed at spot-checking points. We also offer the same amount of security. Cheating drivers are detected, and a dishonest central authority cannot learn any more information than they do by spot-checking.

8.2. Ecash

Chaum [23] introduced the concept of ecash. The essential idea is that a user can purchase a single interaction with a server the end result of which is a cryptographic *coin* in the possession of the user which can only be created by the server. Presuming the server demands a fixed amount of a given currency to create a coin and presuming a coin can be redeemed for the amount in question, the coin may be treated as though it were the underlying currency.

The attractiveness of Chaum's scheme lies in the privacy guarantees. In particular, given two coins created in two separate interactions with the server, it is information-theoretically impossible to relate the coins to the respective events which created each coin.

We present instantiations based on RSA Full Domain Hash (RSA-FDH) and the BLS signature scheme. Chaum's scheme consists of the algorithms `Generate`, `Commit`, `Sign`, `Open`, and `Verify`.

The algorithms `Generate` and `Sign` are run by the signer, who holds the private key, while the remaining algorithms are run by the client, who holds the public key.

8.2.1. RSA-FDH-based ecash. The follow ecash scheme is based on RSA-FDH signatures. In particular, the `Generate` and `Verify` algorithms for the RSA-FDH-based ecash are the same as in RSA-FDH signatures.

`Generate` returns a public key (e, N), a private key (d, N), and a hash function $H\colon \{0,1\}^* \to \mathbb{Z}_N$.

`Commit` takes as input a message m. The algorithm selects $x \in \mathbb{Z}_N$ uniformly at random and returns $c = x^e \cdot H(m)$.

`Sign` takes as input a commitment c and returns $\gamma = c^d$.

`Open` takes as input a signed commitment γ and the corresponding random value x. The algorithm returns the signature $\sigma = \gamma \cdot x^{-1}$.

`Verify` takes as input a public key (e, N) and a message/signature pair (m, σ). The algorithm returns 'valid' if $\sigma^e = H(m)$ and 'invalid' otherwise.

The security of Chaum's scheme can be proven equivalent to solving the One-More-RSA-Inversion problem [12]. In practice, it is generally assumed that this is equivalent to the RSA problem but currently it is only known that the One-More-RSA-Inversion problem can be reduced to the RSA problem.

8.2.2. Pairing-based ecash. Fix a pairing $e\colon G_1 \times G_2 \to G_T$, where G_1, G_2, and G_T are order-r groups. As global public parameters we fix generators $P \in G_1$, $Q \in G_2$, and a hash function $H\colon \{0,1\}^* \to G_1$.

`Generate` selects a positive integer $x < r$ uniformly at random. The algorithm returns a public key $\langle X_1, X_2\rangle = \langle xP, xQ\rangle$ and a private key x.

`Commit` takes as input a message m. The algorithm selects a positive integer $s < r$ uniformly at random and returns a commitment $R = sP + H(m)$.

`Sign` takes as input a commitment R and returns $\gamma = xR$.

`Open` takes as input a signed commitment γ, the corresponding random value s, and a public key $\langle X_1, X_2\rangle$. The algorithm returns the signature $\sigma = \gamma - sX_1$.

`Verify` takes as input a public key $\langle X_1, X_2\rangle$, a message/signature pair (m, σ), and computes $S = H(m)$. The algorithm returns 'valid' if $e(S, X_2) = e(\sigma, P)$ and 'invalid' otherwise.

To save space, the y-coordinate of the signature σ can be discarded; unlike the generic situation for elliptic curve point compression, for BLS, not even one bit need be kept [69]. This comes at a slight extra cost for verification, as that coordinate will need to be recomputed, but this extra cost is negligible compared to the pairing operations (over two orders of magnitude cheaper).

8.3. Spot-record scheme

There are four main components of a tolling scheme. A *registration server* registers drivers, possibly providing them with an on-board unit. A *driver* drives on toll roads and verifiably pays for driving either during or after transit. A *verifier* occasionally monitors the drivers as they drive. A *payment server* ensures that drivers pay for the amount of driving they perform. We could expect that the registration server, payment server, and verifier are managed by the same entity. The goal is to protect the privacy of the driver while enforcing that the driver correctly pays for his driving. Specifically, we concern ourselves with *location privacy*, which is the linkability of the times and places a particular driver has travelled.

In our scheme, the driver obtains *tokens* from the registration server at the beginning of some time period (for example, a month). While driving, the driver spends the tokens by broadcasting them. At the end of the time period, the driver interacts with the payment server to redeem unused tokens and pays for the tokens used.

We present SPEcTRe using the RSA Full Domain Hash signature scheme. However, the BLS signature scheme can be used as well. The tolling protocol consists of three phases: *registration*, *driving*, and *reconciliation*. During registration, the driver is assigned a random private identity i and a set of tokens of the form (r, σ), where $\sigma = H(r, i)^d$ and r is a random string. The identity i should correspond to the license plate of the driver's vehicle but the correspondence should be kept private from third parties.

While driving, the driver broadcasts tuples (r, σ), switching tuples at predefined intervals. In order to ensure drivers are broadcasting, a verifier secretly monitors drivers at random locations and times. For each driver, the verifier takes a picture of the driver's licence plate and records all tuples (r, σ) which are being broadcast. No further work needs to be done by the verifier other than recording and storing this data.

For each license-plate photograph, there is a set of records $\{r_j, \sigma_j\}_j$ that were recorded in the vicinity. The *registration server* derives an identity i from the photograph and ensures that $\sigma_j^e = H(r_j, i)$ for some index j, forming a tuple (i, r_j, σ_j) for each photograph. This server also checks for double spending by ensuring no identity uses the same random string r_j in two different locations. If no signature is found for a given photograph or a signature appears twice, it is concluded that the driver is cheating.

Finally, during reconciliation, the driver submits all tuples (r, σ) which were not spent. The payment server verifies that none of these tokens were detected and collects payment for those tokens not submitted. Alternately, full payment for all tokens can be made at registration time and a refund for the submitted tokens can be received during reconciliation.

If the verifier is capable of taking photographs and detecting tokens in a stealthy manner, then the driver must always spend tokens while driving to avoid penalty. Since a private random identity is chosen for each licence plate, tuples cannot be linked to each other without knowledge of the identity. Since the identity is not broadcast as part of the token, a third party cannot determine the identity corresponding to a given tuple. Location data is only collected during spot-checks. This is similar to VPriv, PrETP, and Milo, and is a vast improvement over schemes implemented today, such as E-ZPass.

8.4. No-record scheme

In the preceding scheme, a small amount of location-time information about honest drivers is recorded in order to combat cheating drivers. However, in an ideal privacy setting, we want to collect no information about honest drivers at all, but still be able to detect dishonest drivers. Our no-record (NR) scheme is an exploration of this possibility.

To accomplish this, we require the additional engineering capability that the spot-checking verifiers be able to determine *which* car is broadcasting which token, through the use of directional antennas or triangulation, for example.

Initially, the registration server runs the ecash `Generate` algorithm and gives the public key to all other components in the scheme.

From the perspective of the driver, our scheme consists of three algorithms: an interactive `Create` algorithm, the `Verify` algorithm, and the `Payment` algorithm. Contained with the `Create` algorithm is the collection of ecash algorithms. The driver selects a random string $m \in_R \{0,1\}^*$ and a random integer $x \in_R \mathbb{Z}_N$. The driver sends $c = x^e H(m)$ to the registration server and obtains c^d from the server. From c^d, the driver can compute the signature $\sigma = x^{-1}c^d = H(m)^d$. The `Create` algorithm is described in Figure 8.1. The driver runs the `Create` algorithm n times to get n tokens from the registration server.

$$\begin{array}{lcl}
\textbf{User} & & \textbf{Registration Server} \\
m \in_R \{0,1\}^* & & \\
x \in_R \mathbb{Z}_N & & \\
c = x^e H(m) & \longrightarrow & \mathrm{c} \\
\gamma & \longleftarrow & \gamma = c^d \\
\sigma = \gamma \cdot x^{-1} & &
\end{array}$$

FIGURE 8.1. The creation of one token

While driving, the driver broadcasts tokens (m, σ) at a predetermined rate. The `Verify` algorithm is run by the verifier as these tokens are received. If the `Verify` algorithm indicates that a signature is invalid, or if the driver is not broadcasting tokens at all, the driver is held accountable in some manner *at the time the algorithm is run*. In practice, the verifier will be associated with a camera or police presence which can take appropriate measures.

After a predetermined period of time (eg. monthly), the driver's tokens are invalidated. At this time, the driver interacts with the payment server to redeem his unused tokens and pay for the service. Invalidation can be done by using a different public key (e, N) for each time period.

The spot-record scheme has the advantage that the verifier does not need to do any work online other than photographing license plates and recording tuples. The no-record scheme has the advantage that the amount of private data recorded is reduced but requires a more involved verifier in order to detect malicious activity, as outlined next in Section 8.4.1.

8.4.1. Combating Double Spending. Malicious drivers have a number of opportunities to double spend their tokens. First, a single driver can use the same token more than once while driving. Second, a single driver can spend a token while driving and then attempt to redeem the token during payment. Third, two drivers can use the token at the same time in different locations. An individual verifier can maintain a sorted list of detected tokens and check new tokens against this list to combat double spending locally. If the verifiers periodically synchronize and ultimately report their results to the payment server, the first and second attacks can be combated. However, without constant communication between verifiers, two drivers using the same token at the same time cannot be detected.

In light of the fact that we need the verifiers to be synchronized to prevent double spending, we introduce a centralized server called the *verify server* which interacts with the verifier to combat double spending. After the verifier ensures the signature is correct, the message m is sent to the verify server to be checked against other messages. The server indicates to the verifier if the token was double spent or not.

8.4.2. Reducing the Cost of Double-Spending Detection. Since verification is a very time-sensitive procedure, we would like to reduce the server-side latency as much as possible. We can search and insert into a B+ tree in time $\mathrm{O}(\log n)$. With even a modest number of vehicles, the number of tokens and verification percentage results in very rapid growth in the size of the B+ tree. If n were small enough to fit in memory, it would greatly reduce the server-side latency and eliminate the need for complicated database-management techniques. To this end, we show that by binding m to a time or location, the double-spending detection cost can be reduced significantly.

We modify SPEcTRe as follows. During the `Register` algorithm, for each possible time t, the driver constructs $m = (H(r,t), H(i,s))$, where r and s are fixed-length bitstrings chosen uniformly at random. The driver obtains a signature on m in the same manner as in the preceeding description of the algorithm with the final result being a set of tuples of the form (t, r, i, s, σ). We suggest that the granularity of t be 5 minutes for instance. If this granularity is too small, then as we will see in Section 8.7 the performance of this scheme might suffer as the registration phase would turn out to be a bottleneck.

While driving, the driver selects the token (t, r, i, s, σ) corresponding to the current time. The driver computes $b = H(i,s)$ and broadcasts (t, r, b, σ). The verifier receives this tuple and rejects the token if t does not correspond to the current time window. It computes the value $m = (H(r,t), b)$ and verifies the signature as $H(m) \overset{?}{=} \sigma^e$. Finally, the value m is sent to a central server to ensure that the token is not being double spent.

During the time corresponding to t, the verify server needs to compare tuples (t, r, b, σ) only to other tuples of the form (t, r', b', σ') in order to detect double spending. If n cars are being detected during the time period corresponding to t, the total time and space complexities for detecting double spending are $\mathrm{O}(n \log n)$ and $\mathrm{O}(n)$ respectively. Of particular note is the amount of space required. One million vehicles and a 160-bit value for m gives a total memory cost of 20 MB.

Finally, the payment server collects all sorted time-based lists and merges them into a master sorted list. The driver computes $a = H(t,r)$ and submits a sorted list of tokens to be redeemed of the form (a, s, σ). Since the identity i is fixed for all tuples, there is no need to submit it multiple times. The payment server computes $m = H(a, H(i,s))$ for each tuple, ensures the list is sorted by the m, runs the `Verify` algorithm on each tuple

	`Verify` / driving	`Payment` / reconciliation	Security assumptions	Privacy	RCD	RVC
VPriv	$O(k)$	$O(k)$	RSA, ZKP, SMC	Partial	No	No
PrETP	$O(k)$	$O(k + u \cdot r)$	RSA, ZKP	Partial	No	No
Milo	$O(k)$	$O(k)$	RSA, ZKP, SMC	Partial	No	No
spot-record	$O(r)$	$O(n \log(n))$	RSA, Hash	Partial	No	No
Basic NR	$O(r \log(u \cdot r))$	$O((n-k) \log(u \cdot r))$	RSA, Hash	Full	Yes	Yes
t-bound NR	$O(r \cdot \rho \log(u \cdot r \cdot \rho))$	$O((n-k) \log(u \cdot r))$	RSA, Hash	Full	Yes	Yes

TABLE 8.1. A comparison of various schemes in terms of the complexities of each phase, security and privacy properties, and also whether real-time cheating detection (RCD) and real-time verifier communications (RVC) are required when drivers are driving on the road. n is the number of tokens per driver. u is the number of drivers. k is the average number of tokens spent per driver. r is the average number of detections per driver. ρ is the granularity of the bound time (a small fraction), so ρ^{-1} is the number of time slices for one month. All algorithms are per driver. ZKP stands for zero-knowledge proofs and SMC denotes secure-multiparty computation. Hash denotes the assumption of a preimage-resistant and collision-resistant hash function. Milo also includes an `Audit` phase which has been grouped with `Verify` (and thus unchanged, since $r < k$). Also note that the constants hidden behind the O notation for schemes with ZKP and SMC are quite large, while for SPEcTRe, they are very small. Privacy means the extent to which data of honest users are unmonitored.

and ensures that there are no repeated tuples. Finally, the payment server checks the submitted list against the master list to detect if the driver is attempting to redeem spent tokens. Since the tokens that drivers are attempting to redeem are bound to their identity, they cannot be spent by two drivers. Hence, the submitted list need not be added to the master list. The driver pays for those tokens not returned to the payment server.

We will describe the security and privacy implications of this modification in the next section. We also remark that the entire m need not be transmitted during the verify phase; we could transmit only the first few bits of $H(m)$. For example, if we expect to store 2^s tokens, if we sent only the first $2s + 60$ bits of $H(m)$, we would experience collisions with probability 2^{-30}.

8.4.3. Complexity. In Table 8.1, we give the complexities of our no-record scheme with and without time-bound tokens. The registration algorithms have identical complexity but the complexities of both `Verify` and `Payment` are greatly reduced with the addition of the time variable. In both algorithms, the bottleneck of computation is on the server side.

8.4.4. Probability of Detecting Double Spending. One of the most attractive aspects of SPEcTRe is that verifiers need not check all tokens. Presuming the drivers cannot detect the verifiers, they have no choice but to assume someone is listening and behave honestly. Let ν denote the fraction of spent tokens which are being detected by verifiers. Then the probability of detecting double spending during verification is ν^2. The probability of detecting a driver attempting to redeem a spent token is ν. Thus, it is important to ensure the penalty for double spending outweighs the benefit.

8.4.5. Security of Tokens. We require that it is difficult for drivers to create more tokens than are given to them during the registration protocol. The security of SPEcTRe is equivalent to the security of ecash, relying on the difficultly of the One-More-RSA-Inversion problem.

8.4.6. Privacy. During registration, it is clear that tokens are private in the information-theoretic sense since they are based on a standard ecash scheme. We argue that no relation can be ascertained between tokens revealed during different algorithms. In particular, for $m = H(a, b)$ with $a = H(t, r)$ and $b = H(i, s)$, since the images a and b are from a salted hash, it is computationally infeasible to relate a to t or b to i.

8.5. Comparison with other work

Table 8.1 shows the complexity of our schemes with various others. Compared to previous work, our schemes rely on much more common security assumptions, such as the RSA problem and random oracles. Although the complexities of payment and reconciliation may appear much more expensive than previous work, our schemes are based on simpler primitives, and so the constants in the big-O notation are much smaller in our scheme, as evidenced by our measurements in Section 8.7.

8.6. Hardware infrastructure

There are three basic hardware components in the verifying and payment phases of SPEcTRe: a transponder on each vehicle, token readers at various points along the road, and a centralized server that maintains the database to check double spending.

Commercial manufacturers are working on transponder devices which allow for reliable vehicle-to-roadside or vehicle-to-vehicle communications. The communication protocol is layered [49], giving application-level developers flexibility in designing and implementing their own protocols. These communications are based on dedicated short-range communications [67]. We did a proof-of-concept experiment, broadcasting tokens in a moving vehicle driving at 40 km/h from a Nexus One smartphone, and they were reliably received on the roadside by another Nexus One.

Token readers could either be police cars moving around discreetly or some reader device on the roadside. Whatever those readers are, it is important that drivers are unable to determine if someone is monitoring their broadcasts.

The registration phase requires some server to sign tokens for clients so that transponders can broadcast these tokens. These operations can be carried out between a driver's personal computer and a server. A simple TLS-protected connection is sufficient.

8.7. Performance measurement

In this section we evaluate the runtime and storage requirements for the primitives of SPEcTRe. In our measurement, we let each client serially go through the three phases: registration, verification, and payment. The measurement is based on the no-record scheme, but with an extreme case where all the tokens broadcast are collected.

The proof-of-concept implementation was written in C++ with a multithreaded registration phase. The code is based on a server-client model where each client, representing a vehicle, first gets tokens for the whole month, reveals half of the tokens, and then uses another half in the payment phase. For every token a client reveals, the server stores that token in the database to check double spending later. Our measurement does not utilize the knowledge of i to speed up the payment phase for simplicity, and we add every token in the payment phase to the database, which actually reduces the speed, because with identities revealed in the payment phase, we can simply check double redemption for each individual client. However, we still found that we beat VPriv in terms of computation time given the same number of tokens used.

We only compare in detail the performance of our spot-record scheme with VPriv, because these two schemes share a similar hardware infrastructure. PrETP requires heavy real-time computation on the on-board unit, which would require a specialized cryptographic coprocessor when the key size goes to 2048 bits. PrETP also requires the incorporation of GPS data on each vehicle. In terms of computation requirements on the server side, for a segment size of 1 km and a key size of 2048 bits, PrETP takes 88.050 seconds to verify the payment of each individual driver [4]. SPEcTRe takes only around 16 seconds to both register and verify one month's worth of tokens for a security level of 128 bits by using pairing-based scheme described earlier, corresponding to an RSA key size of 3072 bits.

We used the C/C++ interface of SQLite to manage our database and OpenSSL libraries to implement the cryptographic primitives. The measurement was run on a laptop with two 1.86 GHz cores and 2 GB RAM, with 32-bit 10.04 Ubuntu (lucid) installed. The client and the server were running on the same machine, so we were not simulating the network. However, we did estimate the communication cost and we will argue that our bandwidth requirement is not high.

Figure 8.2 shows our runtime for three phases when the RSA key size is 1024 bits, corresponding to different number of tokens used. Let us assume the average driver travels 18,000 km per year [72]. This equates to 40 hours of driving per month for each car. If drivers are required to reveal a new token every minute, and tokens are not bound to t,

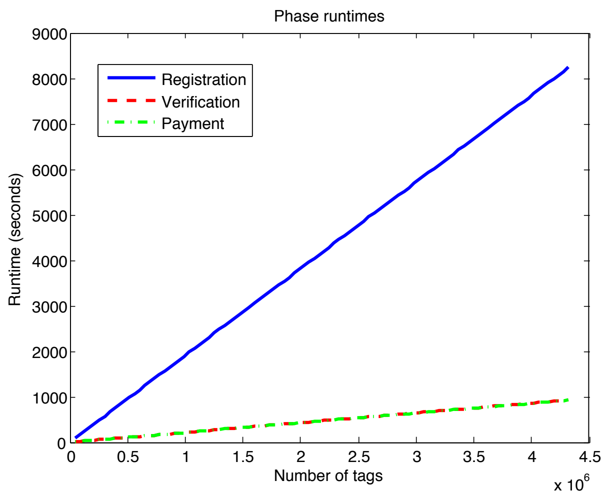

FIGURE 8.2. Runtimes of each phase for many tokens

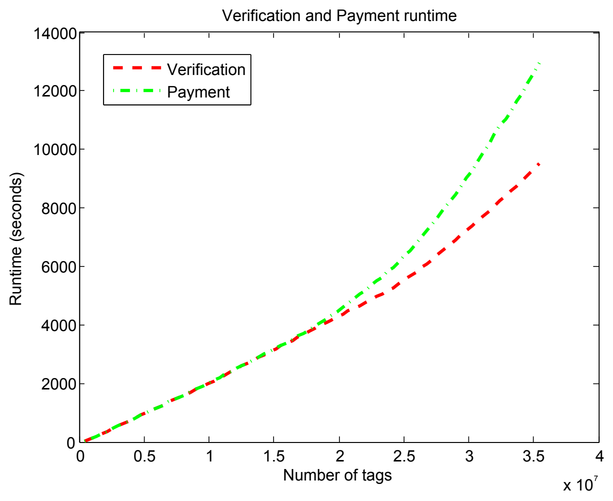

FIGURE 8.3. Runtimes of verification and payment with 10 times as many tokens as Figure 8.2.

then the number of tokens for each client thus would be $40 \times 60 = 2400$ every month. The decision to reveal a new token once per minute was chosen to better compare with VPriv. If we are to bind t to each token, the resolution of time should be chosen carefully so that the number of tokens issued would allow registration to finish in a reasonable amount of time.

There are some practical concerns here, however, some of which make SPEcTRe faster, while some of them make SPEcTRe slower.

One concern is that although the runtime for revealing and payment appears to be linear when the dataset is small, it should not always be the case when the database becomes larger. According to the observation of our experiment, SQLite organizes the database with hash tables. As the database becomes larger the linear performance would not be maintained due to collisions and reorganizations. In order to find out the performance for a larger dataset, we ran another measurement which generates random numbers and insert them into the database. The performance is shown in Figure 8.3, which shows that when the number of tokens grows to 34,992,000 (14,580 clients), the runtime for each insertion grows to twice as long as before.

There are also ways to make SPEcTRe faster. First, we can replace the RSA scheme with a pairing-based scheme. Since currently the bottleneck is the registration phase, and in particular the blind signature computation, a pairing-based scheme would significantly improve our runtime. This is discussed in more detail in Section 8.7.1. Second, the registration phase is also completely parallelizable and can be done in advance. For example, we may choose to allow registration for the next month. All that we need to ensure is that the server's key pair is regenerated every month. Third, we can also easily utilize the i value so that we do not need to add tokens to the database during the payment phase since we only need to check that every client does not submit two identical tokens.

In order to hold the number of tokens generated in our experiment, namely 34,992,000 tokens, the size of the database is 544.9 MB. So in order to hold tokens for one million clients, we need $544.9 \times 1,000,000/14,580 = 37373.11$ MB, which is about 37 GB storage space. This is small for an average server. We can also save some space by not storing the entire token, but instead truncating the hash to a smaller size that makes collisions unlikely, as discussed earlier.

We also measure the communication cost. For the payment phase, the client will reveal every token that is not spent. Assume that a client does not spend any tokens at all, which is the worst case. With the RSA scheme, every token is $|H(t)| + |\sigma| = 256 + 3072 = 3328$ bits at a 128-bit security level. Thus, one client needs to transmit $2,400 \times 3328$ bits $=$ 998 KB in a whole month. For the server, assume that there are 1 million clients who do not spend any tokens at all throughout the whole month. There will be 1 million $\times$ 998 KB which is approximates 998 GB of data transmitted. The bandwidth required for the server is only 0.385 MB/s. If we are to allow multiple prices by requiring more tokens to be revealed on expensive roads, these numbers would go up but they are still acceptable even if multiplied by 10. The pairing-based scheme with a security level of 128 bits has a signature length of 256 bits. In that case, the communication cost would be even less. The communication cost between verify servers is negligible.

We summed up the runtime for all three phases in Figure 8.4, which also compares SPEcTRe with VPriv. For 1800 clients (4,320,000 tokens), SPEcTRe takes about 10,000 seconds. The runtime for VPriv grows linearly with the number of tokens, and VPriv requires 100 seconds for 2,000 tokens in a recommended setting of 10 rounds computation.

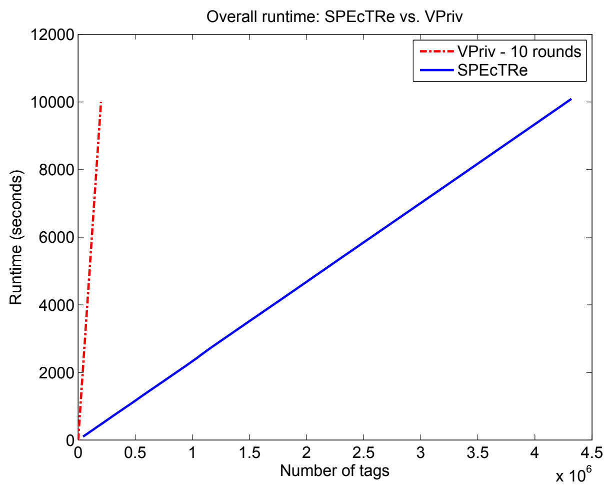

FIGURE 8.4. Runtime comparison of our no-record scheme with VPriv.

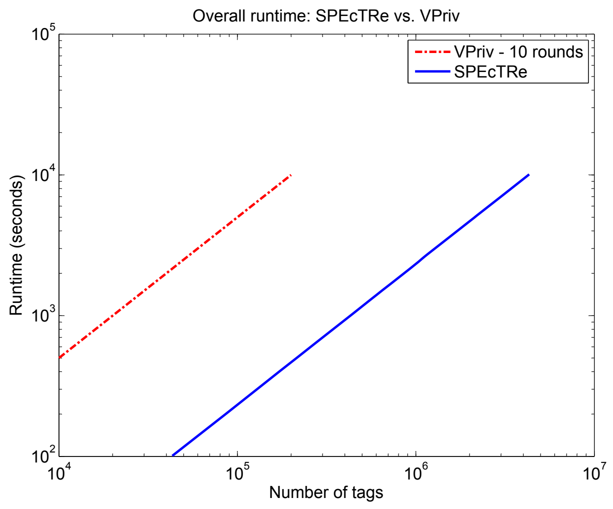

FIGURE 8.5. Log-runtime comparison of our no-record scheme with VPriv.

In this sense, SPEcTRe is $4,320,000/(2,000 \times 100) = 21.6$ times faster. Figure 8.5 better demonstrates this speed difference. A subsequent experiment with a key size of 3072 bits on the 64-bit machine mentioned in the next section yielded a runtime 3 times slower, but still SPEcTRe runs much faster than VPriv. With a pairing-based signature scheme, SPEcTRe runs even faster.

8.7.1. BLS vs RSA. Registration is the slowest phase in the above measurement, as seen in Figure 8.2. By using a pairing-based signature scheme to sign and verify tokens, we are able to make the registration phase much faster. The performance of the BLS pairing-based scheme and the RSA scheme were measured on a 64-bit machine. We used relic-toolkit version 0.3.0 [1] to implement the pairing-based scheme, and OpenSSL to implement the RSA scheme, both at a 128-bit security level (256-bit curves for BLS and a 3072-bit modulus for RSA). With the computation power of a single core, the times are shown in Table 8.2.

Table 8.2. The performance of the BLS and RSA signature schemes

	Sign	Verify	Signature size
BLS	0.4 ms	5.2 ms	256 bits
RSA	13 ms	0.3 ms	3072 bits

As we can see, the pairing-based scheme is doing a much better job in signing blind signatures, while the RSA scheme is slow in signing but much faster in verification. In our no-record scheme, where drivers who are double-spending should be detected on the fly, faster verification reduces one possible bottleneck. However, one should also notice that even assuming 20 cars are passing by within one second, BLS verification only takes 104 ms, which may not be a great issue when compared to network delay. In our spot-record scheme, where double-spending detection can be done offline, verification time is a less critical factor. Given that a client doesn't reveal all his spent tokens to the verifier, it is optimal to utilize faster BLS signing to decrease the overall computation tasks on the server side. The pairing-based scheme requires a smaller signature, which lowers the communication cost at places where signatures are required. But this feature would not help much for the real-time communications between verifiers, because they do not need signatures to be exchanged in order to combat double-spending.

CHAPTER 9

Pairing Inversion

At CRYPTO 2000, Lenstra and Verheul [**54**] presented XTR, a discrete-log public-key cryptosystem which operates in an order-r subgroup X of the order-(p^2-p+1) cyclotomic subgroup of $\mathbb{F}_{p^6}^*$; here p is a prime. XTR was claimed to be as efficient as elliptic curve cryptography, but without being affected by the uncertainty that was still marring the security of elliptic curve cryptography at the time. At the Rump Session of CRYPTO 2000, Menezes and Vanstone [**60**] observed that there is a supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$ of embedding degree 3 with $\#E(\mathbb{F}_{p^2}) = p^2 - p + 1$. Thus, the Weil and Tate pairings yield an efficiently-computable isomorphism $\xi : G \to X$, where G is the order-r subgroup of $E(\mathbb{F}_{p^2})$. Menezes and Vanstone asked about the existence of an efficiently-computable isomorphism $\phi : X \to G$, which would establish the equivalence of DL_G and DL_X, where DL_H denotes the discrete logarithm problem in some group H. However, Verheul [**88**] gave some evidence that such an isomorphism ϕ was unlikely to exist. More generally, Verheul proved that if there exists an efficiently-computable isomorphism $\phi : G_T \to G$, where $e : G \times G \to G_T$ is a symmetric pairing, then CDH_G and CDH_{G_T} can be efficiently solved. Here, CDH_G is the computational Diffie-Hellman problem in G: given $P, aP, bP \in G$, compute abP; CDH_{G_T} is analogously defined. Verheul's theorem is striking because the only method known for solving the Diffie-Hellman problem in a group is to first solve the discrete logarithm problem in that group.

Verheul's theorem has been generalized to other symmetric pairings including those derived from supersingular elliptic curves E defined over $\mathbb{F}_{q^2}$ where $E(\mathbb{F}_{q^2})$ is the product of two cyclic groups of order $q-1$ (such elliptic curves can be considered to have embedding degree $k = \frac{1}{2}$) [**64**], and for certain ordinary elliptic curves with embedding degree $k = 1$ [**44**].

For asymmetric pairings $e : G_1 \times G_2 \to G_T$, it is natural to generalize Verheul's work and ask about the cryptographic implications of the existence of efficiently-computable isomorphisms $\phi_1 : G_T \to G_1$ and $\phi_2 : G_T \to G_2$. Several papers have analyzed specific algorithms for inverting isomorphisms $\phi : G_T \to G$ for symmetric pairings and isomorphisms ϕ_1 and ϕ_2 for asymmetric pairings [**33, 74, 75, 76**]. All these algorithms are inefficient, providing some evidence that Verheul isomorphisms ϕ, ϕ_1 and ϕ_2 cannot be feasibly computed. Nonetheless, these works do not rule out the possibility that efficient algorithms may exist. Since efficiently-computable Verheul isomorphisms can have devastating consequences for the security of pairing-based protocols (e.g., see Remark 9.1), it is important to collect further evidence for their non-existence.

In [34], Galbraith, Hess and Vercauteren studied the implications of the existence of efficient algorithms for certain pairing inversion problems: (i) given $R \in G_1$ and $z \in G_T$, find $S \in G_2$ with $e(R,S) = z$; and (ii) given $S \in G_2$ and $z \in G_T$, find $R \in G_1$ with $e(R,S) = z$. The results in [34] are purported to be generalizations and refinements of Verheul's theorem to asymmetric pairings. However, a strict generalization of Verheul's theorem would be concerned with efficiently-computable isomorphisms ϕ_1 from G_T to G_1 and ϕ_2 from G_T to G_2. In this chapter, we explore the implications of the existence of such isomorphisms.

The contributions of this chapter are the following:

(1) In §9.1 we provide a short proof of Verheul's theorem by establishing the equivalence of a certain fixed argument pairing inversion problem and the problem of computing an isomorphism $\phi : G_T \to G$. Our proof is considerably simpler than Verheul's original proof (see Lemma 2, Lemma 3 and Theorem 4 of [88]) which, among other things, determines the equivalence of three variants of the Diffie-Hellman problem.
(2) In §9.1 we argue that efficiently-computable reductions from DL_{G_T} to DL_G are more general than efficiently-computable isomorphisms from G_T to G. Consequently, Verheul's theorem can be viewed as providing somewhat limited evidence that DL_{G_T} is harder than DL_G.
(3) In §9.2 we show that the existence of efficiently-computable isomorphisms $\phi_1 : G_T \to G_1$ and $\phi_2 : G_T \to G_2$ imply that CDH_{G_1}, CDH_{G_2}, and CDH_{G_T} can be efficiently solved.
(4) In §9.2 we show that the existence of an efficiently-computable isomorphism $\phi : G_T \to G_1$ (resp. $\phi : G_T \to G_2$) implies that DL_{G_2} (resp. DL_{G_1}) can be solved in 'Cheon time' which, under suitable conditions, is optimized to $\tilde{O}(r^{1/3})$. This is significantly faster than the running time $\tilde{O}(\sqrt{r})$ required by Pollard's rho algorithm [71].

9.1. Symmetric pairings

Let $e : G \times G \to G_T$ be a symmetric pairing. An isomorphism $\phi : G_T \to G$ is defined by its action on some generator $g \in G_T$; say $\phi(g) = P$. The *Compute-ϕ* problem is the following: given $z \in G_T$, compute $\phi(z)$. The *Fixed Argument Pairing Inversion (FAPI)* problem is the following: given $R \in G$ and $z \in G_T$, determine $S \in G$ such that $e(R,S) = z$.

Lemma 9.1. The FAPI and Compute-ϕ problems are computationally equivalent.

Proof. The proof that Compute-ϕ reduces to FAPI is straightforward. Suppose that we are given $z = g^\ell$ and a FAPI oracle; we wish to compute $\phi(z) = \ell P$. We first use the FAPI oracle to find $Q \in G$ such that $e(P,Q) = g$, and then use the FAPI oracle to find $R \in G$ for which $e(Q,R) = z$; note that $R = \ell P$.

We now prove that FAPI reduces to Compute-ϕ. So, given $R \in G$, $z \in G_T$, and an oracle for Compute-ϕ, we wish to compute $S \in G$ with $e(R,S) = z$. Let $R = aP$, $S = bP$, $e(P,P) = g^c$, and $z = g^t$. Since $e(aP, bP) = g^t$, we have $bP = a^{-1}c^{-1}tP$. Now, define $T(i) = a^i c^{i-1} P$ for $i \geq 1$, and note that $T(1) = R$. Given $T(i)$, one can efficiently compute $T(2i)$ since

$$e(T(i), T(i)) = e(a^i c^{i-1} P, a^i c^{i-1} P) = g^{a^{2i} c^{2i-1}}$$

and $\phi(g^{a^{2i}c^{2i-1}}) = a^{2i}c^{2i-1}P = T(2i)$. Moreover, given $T(2i)$, one can efficiently compute $T(2i+1)$ since

$$e(T(2i), T(1)) = e(a^{2i}c^{2i-1}P, aP) = g^{a^{2i+1}c^{2i}}$$

and $\phi(g^{a^{2i+1}c^{2i}}) = a^{2i+1}c^{2i}P = T(2i+1)$. Thus, by processing the bits of the binary representation of $r-2$ from left to right, one can use a double-and-add strategy to efficiently compute

$$T(r-2) = a^{r-2}c^{r-3}P = a^{-1}c^{-2}P.$$

Finally, one can efficiently compute $\phi(g^t) = tP$, $e(tP, a^{-1}c^{-2}P) = g^{a^{-1}c^{-1}t}$ and $\phi(g^{a^{-1}c^{-1}t}) = a^{-1}c^{-1}tP = S$. □

Lemma 9.1 immediately gives a short proof of Verheul's Theorem.

Theorem 9.2 (Verheul)**.** Let $\phi : G_T \to G$ be an isomorphism defined by $\phi(g) = P$ and suppose that ϕ can be efficiently computed. Then CDH_G and CDH_{G_T} can be efficiently solved.

Proof. By Lemma 9.1, we can efficiently solve the FAPI problem.

Suppose that we are given a CDH_G instance (Q, aQ, bQ) and wish to compute abQ. We compute $z = e(aQ, bQ)$ and then find $R \in G$ such that $e(Q,R) = z$; note that $R = abQ$.

Suppose now that we are given a CDH_{G_T} instance (h, h^x, h^y) and wish to compute h^{xy}. We do the following:

(i) Find $Q \in G$ with $e(P,Q) = h$.
(ii) Find $R \in G$ with $e(Q,R) = h^x$; note that $R = xP$.
(iii) Find $S \in G$ with $e(P,S) = h^y$; note that $S = yQ$.
(iv) Compute $e(R,S) = e(xP, yQ) = h^{xy}$.

□

The point of Verheul's theorem is to argue that DL_{G_T} cannot be reduced to DL_G, thus providing evidence that DL_{G_T} is harder than DL_G. However, as observed in [50], Verheul's theorem can also be viewed as having negative consequences for pairing-based cryptography wherein someone who mistrusts the security of elliptic curve cryptosystems may have their worries allayed by the assurance that DL_G is no easier than DL_{G_T}.

A more relevant question is whether there exists a *reduction* of DL_{G_T} to DL_G. Such a reduction is an algorithm $\mathcal{R}$ which on input $h, h^x \in G_T$ and an oracle for solving DL_G, computes x. Clearly, an algorithm for computing some isomorphism $\phi : G_T \to G$ is also a reduction of DL_{G_T} to DL_G. The interesting question is whether *every* reduction of DL_{G_T} to DL_G in fact yields an efficiently-computable isomorphism from G_T to G. By Lemma 9.1, an equivalent question is whether every reduction of DL_{G_T} to DL_G yields an efficient FAPI solver:

Question 9.1. Is there an algorithm $\mathcal{A}$ which, when given oracle access to a reduction algorithm $\mathcal{R}$ and inputs $z \in G_T$, $R \in G$, outputs $S \in G$ such that $e(R, S) = z$?

Suppose there is an efficient algorithm $\mathcal{A}$ which solves the problem posed in Question 9.1 when given *black-box* access to $\mathcal{R}$[1]. Then $\mathcal{A}$ can be used to efficiently solve the following mixed FAPI-DL_G problem: given $z \in G_T$, $R \in G$, $U \in G$, $xU \in G$, compute x or $S \in G$ with $e(R, S) = z$. Namely, given a FAPI-DL_G problem instance (z, R, U, xU), we invoke algorithm $\mathcal{A}$ with inputs (z, R). If $\mathcal{A}$ does not make any calls to its oracle $\mathcal{R}$, then $\mathcal{A}$ outputs S which solves the FAPI-DL_G instance. If $\mathcal{A}$ makes a call to $\mathcal{R}$, then we (in our role as simulator for $\mathcal{R}$), request the solution of the DL_G instance (U, xU); since $\mathcal{A}$ is responsible for answering $\mathcal{R}$'s oracle queries, $\mathcal{A}$ returns x which again solves the FAPI-DL_G instance.

Since FAPI-DL_G is expected to be intractable, the above argument suggests that reductions of DL_{G_T} to DL_G are in fact *more general* than efficiently-computable isomorphisms from G_T to G. Hence, Verheul's theorem can be viewed as providing somewhat limited evidence that DL_{G_T} is harder than DL_G since it does not fully address the question of whether there is a (general) *reduction* from DL_{G_T} to DL_G.

9.2. Asymmetric pairings

Let $e : G_1 \times G_2 \to G_T$ be an asymmetric pairing, where G_1, G_2, G_T are groups of prime order r and $G_1 \neq G_2$. Furthermore, let g be a fixed generator of G_T. An isomorphism $\phi_1 : G_T \to G_1$ is defined by its action on g; say $\phi_1(g) = P_1$. Similarly, an isomorphism $\phi_2 : G_T \to G_2$ is defined by its action on g; say $\phi_2(g) = P_2$. The *Compute-ϕ_1* problem is the following: given $z \in G_T$, compute $\phi_1(z)$. The *Compute-ϕ_2* problem is the following: given $z \in G_T$, compute $\phi_2(z)$. The *FAPI-1* problem is the following: given $R \in G_1$ and $z \in G_T$, determine $S \in G_2$ with $e(R, S) = z$. Similarly, the *FAPI-2* problem is the following: given $S \in G_2$ and $z \in G_T$, determine $R \in G_1$ with $e(R, S) = z$.

[1]We have nothing useful to say in the case where $\mathcal{A}$ is given non-black-box access to $\mathcal{R}$.

Galbraith, Hess and Vercauteren [34] proved the following:

Theorem 9.3.

(i) Suppose that FAPI-1 and FAPI-2 can both be efficiently solved. Then CDH_{G_1}, CDH_{G_2} and CDH_{G_T} can be efficiently solved.

(ii) Suppose that FAPI-1 can be efficiently solved, and suppose that we have an efficiently-computable isomorphism $\psi_2 : G_2 \to G_1$. Then FAPI-2 can be efficiently solved.

(iii) Suppose that FAPI-2 can be efficiently solved, and suppose that we have an efficiently-computable isomorphism $\psi_1 : G_1 \to G_2$. Then FAPI-1 can be efficiently solved.

Suppose that FAPI-2 can be efficiently solved. Then one can easily construct an efficiently-computable isomorphism $\phi_1 : G_T \to G_1$ — select arbitrary $S \in G_2$ and $g \in G_T$ and define ϕ_1 by $g \mapsto P$ where $e(P, S) = g$; then $\phi_1(z) = R$ where $e(R, S) = z$. However, it is not known whether an efficient FAPI-2 solver can be constructed from an efficiently-computable isomorphism $\phi_1 : G_T \to G_1$. The next result provides a partial answer to this question.

Theorem 9.4.

(i) Suppose that efficiently-computable isomorphisms $\phi_1 : G_T \to G_1$ and $\psi_1 : G_1 \to G_2$ are known. Then FAPI-2 can be efficiently solved.

(ii) Suppose that efficiently-computable isomorphisms $\phi_2 : G_T \to G_2$ and $\psi_2 : G_2 \to G_1$ are known. Then FAPI-1 can be efficiently solved.

Proof. We prove (i); the proof of (ii) is analogous.

Given $S \in G_2$ and $z \in G_T$, we wish to find $R \in G_1$ with $e(R, S) = z$. Let $\phi_1(z) = aR$ and $\psi_1(R) = bS$ for some (unknown) integers a and b. Define the maps $\alpha : G_1 \to G_1$ and $\beta : G_1 \times G_1 \to G_1$ by

$$\alpha(U) = \phi_1(e(U, S)) \tag{9.1}$$

and

$$\beta(U, V) = \phi_1(e(U, \psi_1(V))). \tag{9.2}$$

Observe that $\alpha(U) = aU$ and $\beta(U, V) = abcdR$ where $U = cR$ and $V = dR$, and that α and β can be efficiently computed.

For notational convenience, we identify a point $a^{i-1}b^{j-1}R$ with the vector $[i, j]$ whose components are integers modulo $r - 1$. Thus,

$$\alpha([i, j]) = [i + 1, j] \tag{9.3}$$

and

$$\beta([i, j], [k, \ell]) = [i + k, j + \ell]. \tag{9.4}$$

Our goal is to efficiently compute $[1,1]$, which corresponds to R. We begin by computing $\phi_1(z) = aR$ which corresponds to $[2,1]$. Using (9.4), one can process the bits of the binary representation of $r-2$ to efficiently compute $(r-2)\cdot[2,1] = [-2,-1]$, followed by $\alpha([-2,-1]) = [-1,-1]$. Finally, one uses (9.4) again to efficiently compute $(r-2)\cdot[-1,-1] = [1,1]$. □

The next result, which can be viewed as a refinement of Verheul's theorem for asymmetric pairings, follows immediately from Theorems 9.3 and 9.4.

Corollary 9.5. (i) Suppose that efficiently-computable isomorphisms $\phi_1 : G_T \to G_1$ and $\psi_1 : G_1 \to G_2$ are known. Then CDH_{G_1}, CDH_{G_2} and CDH_{G_T} can be efficiently solved.

(ii) Suppose that efficiently-computable isomorphisms $\phi_2 : G_T \to G_2$ and $\psi_2 : G_2 \to G_1$ are known. Then CDH_{G_1}, CDH_{G_2} and CDH_{G_T} can be efficiently solved.

Theorem 9.6. Suppose that efficiently-computable isomorphisms $\phi_1 : G_T \to G_1$ and $\phi_2 : G_T \to G_2$ are known. Then FAPI-1 and FAPI-2 can be efficiently solved.

Proof. We show that FAPI-2 can be efficiently solved. Given $S \in G_2$ and $z \in G_T$, we wish to find $R \in G_1$ with $e(R,S) = z$. Let $\phi_1(z) = aR$ and $\phi_2(z) = bS$ for some (unknown) integers a and b. Define the maps $\alpha : G_T \to G_T$ and $\beta : G_T \times G_T \to G_T$ by

$$\alpha(u) = e(\phi_1(u), S)) \tag{9.5}$$

and

$$\beta(u,v) = e(\phi_1(u), \phi_2(v)). \tag{9.6}$$

Observe that $\alpha(u) = u^a$ and $\beta(u,v) = z^{abcd}$ where $u = z^c$ and $v = z^d$, and that α and β can be efficiently computed.

For notational convenience, we identify an element $z^{a^{i-1}b^{j-1}}$ with the vector $[i,j]$ whose components are integers modulo $r-1$. Thus,

$$\alpha([i,j]) = [i+1,j] \tag{9.7}$$

and

$$\beta([i,j],[k,\ell]) = [i+k, j+\ell]. \tag{9.8}$$

We are given the element z, which corresponds to $[1,1]$, and our goal is to efficiently compute $R = \phi_1(z^{a^{-1}})$. We compute $(r-2)\cdot[1,1] = [-1,-1]$, followed by $\alpha([-1,-1]) = [0,-1]$. Finally, we compute $(r-2)\cdot[0,-1] = [0,1]$ which corresponds to $z^{a^{-1}}$ and $\phi_1(z^{a^{-1}})$. □

Theorems 9.3(i) and 9.6 immediately give the following generalization of Verheul's theorem for asymmetric pairings.

Corollary 9.7. Suppose that efficiently-computable isomorphisms $\phi_1 : G_T \to G_1$ and $\phi_2 : G_T \to G_2$ are known. Then CDH_{G_1}, CDH_{G_2} and CDH_{G_T} can be efficiently solved.

Let G be an additively-written group of prime order r, and suppose that $d \mid r-1$. Cheon [25] (see also [20]) showed how, given $P, xP, x^dP \in G$, one can compute x in time

$$O\left(\log r\left(\sqrt{\frac{r}{d}}+\sqrt{d}\right)\right)$$

and memory $O(\max\{\sqrt{r/d}, \sqrt{d}\})$. Note that if $d \approx r^{1/3}$, the running time and memory of Cheon's algorithm is $\tilde{O}(r^{1/3})$, which is faster than the running time $\tilde{O}(\sqrt{r})$ of Pollard's rho algorithm for computing logarithms in G. Morales [65] showed that if FAPI-2 can be efficiently solved for an asymmetric pairing $e : G_1 \times G_2 \to G_T$, then DL_{G_2} can be solved in time

$$O\left(\log r\left(\sqrt{\frac{r}{d}}+\sqrt{d}\right)+d\right). \tag{9.9}$$

For convenience, we will refer to (9.9) as 'Cheon time'. Note that Cheon time and the number of oracle calls are jointly minimized when $d \approx r^{1/3}$.

We present extensions of Morales's result to the situation where an efficiently-computable isomorphism $\phi_1 : G_T \to G_1$ is known. We let P, Q, g denote fixed generators of G_1, G_2, G_T with $e(P,Q) = g$.

Theorem 9.8. Suppose that an efficiently-computable isomorphism $\phi_1 : G_T \to G_1$ is known. Then FAPI-2 can be solved in Cheon time.

Proof. Let $S \in G_2$ and $z \in G_T$ be an instance of the FAPI-2 problem. Let $S = yQ$ and $z = g^{xy}$ for some $x, y \in [1, r-1]$. Our goal is to compute $R = xP$. Let $\phi_1(g) = aP$. Define $z_i = z^{(ay)^i}$ and $P_i = (ay)^i xP$ for $i \geq 0$. Since $z_0 = z$, $P_{i+1} = \phi_1(z_i)$, and $e(P_i, S) = z_i$ for $i \geq 0$, we can iteratively compute $z_1, z_2, \ldots, z_d$ in time $\tilde{O}(d)$. Now, using Cheon's algorithm with input z_i for $i = 0, 1, \ldots, d$, we can compute ay in Cheon time. Finally, we compute $R = (ay)^{-1}P_1 = xP$. □

Theorem 9.8 and Morales's result immediately show, given an efficiently-computable isomorphism $\phi_1 : G_T \to G_1$, that DL_{G_2} can be solved in time $\tilde{O}(\sqrt{rd} + d^2)$. However, this result is not interesting since Pollard's rho algorithm already solves DL_{G_2} in $\tilde{O}(\sqrt{r})$ time. Theorem 9.10 is a useful variant of this result.

Lemma 9.9. Suppose that an efficiently-computable isomorphism $\phi_1 : G_T \to G_1$ is known, and suppose that $\phi_1(g) = aP$. Then the integer a can be computed in Cheon time.

Proof. Let $P_i = a^iP$ and $g_i = g^{a^i}$ for $i \geq 0$. Since $P_1 = aP$, $e(P_i, Q) = g_i$, and $\phi_1(g_i) = P_{i+1}$, we can iteratively compute $g_1, g_2, \ldots, g_d$ in time $\tilde{O}(d)$. Finally, Cheon's algorithm can be used to compute a. □

Theorem 9.10. (i) Suppose that an efficiently-computable isomorphism $\phi_1 : G_T \to G_1$ is known. Then DL_{G_2} can be solved in Cheon time.

(ii) Suppose that an efficiently-computable isomorphism $\phi_2 : G_T \to G_2$ is known. Then DL_{G_1} can be solved in Cheon time.

Proof. We prove (i); the proof of (ii) is analogous.

Let $\phi_1(g) = aP$. Suppose that we are given a DL_{G_2} instance (Q, S); we need to find the modulo-r integer y such that $S = yQ$. Let $P_i = (ay)^i aP$ and $g_i = g^{(ay)^i}$ for $i \geq 0$. Since $P_0 = aP$, $e(P_i, S) = g_{i+1}$, and $\phi_1(g_i) = P_i$, we can iteratively compute $g_1, g_2, \ldots, g_d$ in time $\tilde{O}(d)$. Next, Cheon's algorithm can be used to compute $ay \bmod r$. Finally, a can be computed in Cheon time using Lemma 9.9, and thereafter $y = a^{-1}(ay) \bmod r$ can be immediately computed. □

Remark 9.1. By Theorem 9.10, the existence of either an efficiently-computable isomorphism $\phi_1 : G_T \to G_1$ or an efficiently-computable isomorphism $\phi_2 : G_T \to G_2$ will have damaging consequences to the security of pairing-based protocols. For example, in the BLS signature scheme (cf. Section 1.2.2), an entity's private key is an integer x selected uniformly at random from the interval $[1, r-1]$, and the corresponding public key is $X = xQ$. The entity's signature on a message $m \in \{0,1\}^*$ is $\sigma = xP$, where $P = H(m)$ and $H : \{0,1\}^* \to G_1$ is a hash function. The signed message (m, σ) can be verified by computing $P = H(m)$ and checking that $e(\sigma, Q) = e(P, X)$. If an efficiently-computable isomorphism $\phi_1 : G_T \to G_1$ is known, then the DL_{G_2} instance (Q, X) can be solved in Cheon time to recover the private key x. If an efficiently-computable isomorphism $\phi_2 : G_T \to G_2$ is known then, given a single signed message (m, σ), the DL_{G_1} instance (P, σ) can be solved in Cheon time to determine x.

Remark 9.2. Let $u = -(2^{62} + 2^{55} + 1)$ and consider the BN elliptic curve

$$E : Y^2 = X^3 + 2 \tag{9.10}$$

defined over $\mathbb{F}_p$, where $p = 36u^4 + 36u^3 + 24u^2 + 6u + 1$. This elliptic curve has the property that

$$r = \#E(\mathbb{F}_p) = 36u^4 + 36u^3 + 18u^2 + 6u + 1 \tag{9.11}$$

is a 254-bit prime. One can check that there exists an 85-bit divisor d of $r - 1$ which optimizes Cheon time (9.9). Hence, by Theorem 9.10(i), if there exists an efficiently-computable isomorphism $\phi_1 : G_T \to G_1$, then DL_{G_2} can be solved in roughly 2^{85} time — much faster than the 2^{127} time required by Pollard's rho algorithm.

CHAPTER 10

Future work

In Chapter 3, we discussed the use of automorphisms to reduce the Miller length. Hess [**42**] has observed that endomorphisms can be used reduce the Miller length. For example, a supersingular curve E over $\mathbb{F}_{2^m}$ has an endormophism $\sigma\colon (x, y) \mapsto (x^2, y^2)$ when the curve equation is defined over $\mathbb{F}_2$. For such a curve, we can write the ate pairing as

$$f_{2^m,P}(Q) = \prod_{i=1}^{m-1} f_{2,2^i P}(Q)^{2^{m-1-i}}.$$

Using the fact that squaring a Miller function is equivalent to squaring the coordinates of the input points, we have

$$f_{2^m,P}(Q) = \prod_{i=1}^{m-1} f_{2,2^i P}(Q)^{2^{m-1-i}} = \prod_{i=1}^{m-1} f_{2,2^i\sigma^{m-1-i}P}(\sigma^{m-1-i}Q).$$

By determining the action of σ in relation to point doubling, we can find a method to compute the ate pairing as a product of Miller functions with length 2, evaluated at points whose coordinates are of the form $\sigma^i P$, $\sigma^j Q$.

For the curve E defined by $Y^2 + Y = X^3 + X + b$ over $\mathbb{F}_{2^m}$ where $b \in \{0, 1\}$, Barreto *et al.* [**11**] observed that (a power of) the ate pairing can be computed as

$$f_{2^m,P}(Q) = \prod_{i=1}^{m-1} f_{2,\sigma^{-i}P}(\sigma^i Q).$$

This observation was arrived at by examining the explicit equations used in the double-and-add step in the pairing computation and predates Hess-Vercauteren optimal pairings. It would be interesting to further study these properties of supersingular curves in terms of Hess-Vercauteren optimal pairings and the use of endomorphisms in general.

For elliptic curves, the utilization of automorphisms in pairing computation does not always indicate a clear advantage. For a curve E with embedding degree k and an order-d automorphism group, we may use automorphisms to reduce the length of the Miller loop if d does not divide k. However, in order to obtain an efficient representation of G_2, we desire that d divides k. Since an efficient representation of G_2 is an important consideration when choosing a curve for efficient implementation, curves which would benefit from the use of automorphisms in the pairing computation are often not considered competitive. The Hess-Vercauteren optimal pairing framework applies analogously to hyperelliptic curves, where there exists a wider variety of automorphism groups. There is potentially a huge advantage to be obtained if suitable hyperelliptic curves are discovered.

Many of our optimizations have focused on the computation of a single pairing. In practice, multiple pairing computations are required. For example, a BLS signature scheme where we vary the number of messages and signatures produces some interesting results. Recall that the BLS protocol consists of a public key $X = xP$ and signatures $\sigma = xH(m)$ are generated using the private key x. We can verify the BLS signature σ as

$$e(H(m), X) = e(\sigma, P).$$

Given two processor cores, we can simply give each pairing to each processor core. Given a single processor core, we verify the signature as

$$e(H(m), X) \cdot e(\sigma, -P) = 1.$$

This allows to compute the Miller functions of each pairing first, take their product, and finish with a single final exponentiation, saving a bit of computation. Given four processor cores, we can apply our parallelization results to spread the pairings over two processor cores each.

On the other extreme, suppose we are given messages $(m_1, \ldots, m_N)$ corresponding to signatures $(\sigma_1, \ldots, \sigma_N)$ and signers $(X_1, \ldots, X_N)$. Then we can verify the signatures as

$$\prod_{i=1}^{N} e(H(m_i), X_i)^{c_i} = e\left(\sum_{i=1}^{N} c_i\sigma_i, P\right),$$

where the c_i are chosen uniformly at random. If we have more than four times as many processor cores as we do signers, then our parallel results will reduce the latency of the verification.

If we consider a single signer with multiple signatures, then we can optimize verification as

$$e\left(\sum_{i=1}^{N} c_i H(m), X\right) = e\left(\sum_{i=1}^{N} c_i\sigma_i, P\right)$$

which, as in the single-signer single-message case, we can reduce to

$$e\left(\sum_{i=1}^{N} c_i H(m), X\right) \cdot e\left(\sum_{i=1}^{N} c_i\sigma_i, -P\right) = 1.$$

There are many variables to consider in determining the best method for parallelization, including number of signers, number of messages, security level, and level of precomputation.

References

[1] D. Aranha and C. Gouvêa. RELIC is an efficient library for cryptography. http://code.google.com/p/relic-toolkit/.

[2] D. Aranha, K. Karabina, P. Longa, C. Gebotys, and J. López. Faster explicit formulas for computing pairings over ordinary curves. *Advances in Cryptology–EUROCRYPT 2011*, pages 48–68, 2011.

[3] D. Aranha, J. López, and D. Hankerson. High-speed parallel software implementation of the η_T pairing. *Topics in Cryptology-CT-RSA 2010*, pages 89–105, 2010.

[4] J. Balasch, A. Rial, C. Troncoso, C. Geuens, B. Preneel, and I. Verbauwhede. PrETP: Privacy-preserving electronic toll pricing. In *19th USENIX Security Symposium*, Washington, DC, 2010.

[5] R. Balasubramanian and N. Koblitz. The improbability that an elliptic curve has subexponential discrete log problem under the Menezes–Okamoto–Vanstone algorithm. *Journal of Cryptology*, 11(2):141–145, 1998.

[6] E. Bangerter, E. Camenisch, and A. Lysyanskaya. A cryptographic framework for the controlled release of certified data. In *Security Protocols Workshop*, pages 20–42, 2004.

[7] P. Barreto, H. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. *Advances in Cryptology–CRYPTO 2002*, pages 354–369, 2002.

[8] P. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. *Security in Communication Networks*, pages 257–267, 2003.

[9] P. Barreto, B. Lynn, and M. Scott. On the selection of pairing-friendly groups. In *Selected Areas in Cryptography – SAC 2003*, pages 17–25, 2004.

[10] P. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography – SAC 2005*, pages 319–331, 2006.

[11] P.S.L.M. Barreto, S. Galbraith, C.Ó. hÉigeartaigh, and M. Scott. Efficient pairing computation on supersingular abelian varieties. *Designs, Codes and Cryptography*, 42(3):239–271, 2007.

[12] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16:185–215, 2003.

[13] M. Bellare and P. Rogaway. The exact security of digital signatures-how to sign with RSA and Rabin. In *Advances in Cryptology–Eurocrypt'96*, pages 399–416, 1996.

[14] J.L. Beuchat, J. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over Barreto–Naehrig curves. *Pairing-Based Cryptography – Pairing 2010*, pages 21–39, 2010.

[15] D. Boneh and X. Boyen. Short signatures without random oracles. In *Advances in Cryptology-EUROCRYPT 2004*, pages 56–73, 2004.

[16] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology–CRYPTO 2001*, pages 213–229, 2001.

[17] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. *Advances in Cryptology–EUROCRYPT 2003*, pages 641–641, 2003.

[18] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Advances in Cryptology–ASIACRYPT 2001*, pages 514–532, 2001.

[19] F. Brezing and A. Weng. Elliptic curves suitable for pairing based cryptography. *Designs, Codes and Cryptography*, 37(1):133–141, 2005.

[20] D. Brown and R. Gallant. The static Diffie-Hellman problem. *Centre for Applied Cryptographic Research, University of Waterloo, Technical Report CACR 2004-10*, 2004.

[21] R. Cervero. *The Transit Metropolis: A Global Inquiry*, chapter 7. Transportation / Planning. Island Press, 1998.

[22] S. Chatterjee, D. Hankerson, E. Knapp, and A. Menezes. Comparing two pairing-based aggregate signature schemes. *Designs, Codes and Cryptography*, 55(2):141–167, 2010.

[23] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology – CRYPTO 1982*, pages 199–203, 1982.

[24] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28:1030–1044, October 1985.

[25] J. Cheon. Security analysis of the strong Diffie-Hellman problem. *Advances in Cryptology-EUROCRYPT 2006*, pages 1–11, 2006.

[26] J. Coron. On the exact security of full domain hash. In *Advances in Cryptology – CRYPTO 2000*, pages 229–235, 2000.

[27] C. Costello, T. Lange, and M. Naehrig. Faster pairing computations on curves with high-degree twists. *Public Key Cryptography–PKC 2010*, pages 224–242, 2010.

[28] C. Costello, K. Lauter, and M. Naehrig. Attractive subfamilies of BLS curves for implementing high-security pairings. *Progress in Cryptology–INDOCRYPT 2011*, pages 320–342, 2011.

[29] A. Devegili, M. Scott, and R. Dahab. Implementing cryptographic pairings over Barreto-Naehrig curves. *Pairing-Based Cryptography–Pairing 2007*, pages 197–207, 2007.

[30] D. Freeman. Constructing pairing-friendly elliptic curves with embedding degree 10. *Algorithmic Number Theory*, pages 452–465, 2006.

[31] D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, 2010.

[32] S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. *Algorithmic Number Theory*, pages 69–86, 2002.

[33] S. Galbraith, C.Ó. hÉigeartaigh, and C. Sheedy. Simplified pairing computation and security implications. *Journal of Mathematical Cryptology*, 1(3):267, 2007.

[34] S. Galbraith, F. Hess, and F. Vercauteren. Aspects of pairing inversion. *IEEE Transactions on Information Theory*, 54(12):5719–5728, 2008.

[35] S. Galbraith, X. Lin, and M. Scott. Endomorphisms for faster elliptic curve cryptography on a large class of curves. *Journal of Cryptology*, 24(3):446–469, 2011.

[36] S. Galbraith, K.G. Paterson, and N. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[37] S. Galbraith and M. Scott. Exponentiation in pairing-friendly groups using homomorphisms. *Pairing-Based Cryptography–Pairing 2008*, pages 211–224, 2008.

[38] P. Grabher, J. Großschädl, and D. Page. On software parallel implementation of cryptographic pairings. In *Selected Areas in Cryptography – SAC 2008*, pages 35–50, 2009.

[39] R. Granger and M. Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. *Public Key Cryptography–PKC 2010*, pages 209–223, 2010.

[40] D. Hankerson, A. Menezes, and M. Scott. Software implementation of pairings. *Identity-Based Cryptography*, pages 188–206, 2008.

[41] F. Hess. A note on the Tate pairing of curves over finite fields. *Archiv der Mathematik*, 82(1):28–32, 2004.

[42] F. Hess. Pairing lattices. *Pairing-Based Cryptography–Pairing 2008*, pages 18–38, 2008.

[43] F. Hess, N. Smart, and F. Vercauteren. The eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, 2006.

[44] Z. Hu, M. Xu, and Z. Zhou. A generalization of Verheul's theorem for some ordinary curves. In *Information Security and Cryptology*, pages 105–114, 2011.

[45] P. Hubbard. Urban congestion–why 'free' roads are costly. Treasury Department, Commonwealth of Australia.

[46] S. Ionica and A. Joux. Pairing computation on elliptic curves with efficiently computable endomorphism and small embedding degree. *Pairing-Based Cryptography-Pairing 2010*, pages 435–449, 2010.

[47] E. Kachisa, E. Schaefer, and M. Scott. Constructing Brezing-Weng pairing-friendly elliptic curves using elements in the cyclotomic field. *Pairing-Based Cryptography–Pairing 2008*, pages 126–135, 2008.

[48] K. Karabina. Squaring in cyclotomic subgroups. *Mathematics of Computation*, 82:555–579, 2012.

[49] L. Klein. *Sensor Technologies and Data Requirements for ITS*, chapter 7. Artech House, 2001.

[50] N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. *Cryptography and Coding*, pages 13–36, 2005.

[51] K. Lauter, P. Montgomery, and M. Naehrig. An analysis of affine coordinates for pairing computation. *Pairing-Based Cryptography-Pairing 2010*, pages 1–20, 2010.

[52] E. Lee. The problem with threads. *Computer*, 39(5):33–42, 2006.

[53] A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[54] A. Lenstra and E. Verheul. The XTR public key system. In *Advances in Cryptology–CRYPTO 2000*, pages 1–19, 2000.

[55] C. Lim and P. Lee. More flexible exponentiation with precomputation. In *Advances in Cryptology–CRYPTO 1994*, pages 95–107, 1994.

[56] T. Litman. London congestion pricing: Implications for other cities. http://www.vtpi.org/london.pdf, 2003.

[57] A. Lysyanskaya, R. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Selected Areas in Cryptography – SAC 1999*, pages 184–199, 2000.

[58] S. Meiklejohn, K. Mowery, S. Checkoway, and H. Shacham. The phantom tollbooth: Privacy-preserving electronic toll collection in the presence of driver collusion. In *Proceedings of USENIX Security 2011*, San Francisco, CA, August 2011.

[59] A. Menezes. *Elliptic Curve Public Key Cryptosystems*, volume 234. Kluwer Academic Publishers, 1993.

[60] A. Menezes and S. Vanstone. ECSTR (XTR): Elliptic curve singular trace representation. *Rump Session of Crypto*, 2000, 2000.

[61] V. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.

[62] H. Minkowski. *Geometrie der Zahlen*, volume 1896. Teubner, 1910.

[63] A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E84-A(5):1234–1243, 2001.

[64] D. Moody. The Diffie–Hellman problem and generalization of Verheul's theorem. *Designs, Codes and Cryptography*, 52(3):381–390, 2009.

[65] D. Morales. Cheon's algorithm, pairing inversion and the discrete logarithm problem. Technical report, Cryptology ePrint Archive, Report 2008/300, 2008.

[66] C. Newmarker. Toll records catch unfaithful spouses. http://www.usatoday.com/tech/news/surveillance/2007-08-10-ezpass_N.htm", August 2007.

[67] Dept of Transportation. Dedicated short range communications. http://www.standards.its.dot.gov/Documents/advisories/dsrc_advisory.htm. ITS Standards Advisories No.3.

[68] J. Olivos. On vectorial addition chains. *Journal of Algorithms*, 2(1):13–21, 1981.

[69] D. Page, N. Smart, and F. Vercauteren. A comparison of MNT curves and supersingular curves. *Appl. Algebra Eng., Commun. Comput.*, 17:379–392, October 2006.

[70] G. Pereira, M. Simplício, M. Naehrig, and P. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 84:1319–1326, 2011.

[71] J. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[72] R. Popa, H. Balakrishnan, and A. Blumberg. Vpriv: Protecting privacy in location-based vehicular services. In *18th USENIX Security Symposium*, Montreal, Canada, August 2009.

[73] M. Rieback, B. Crispo, and A. Tanenbaum. The evolution of RFID security. *IEEE Pervasive Computing*, 5:62–69, 2006.

[74] T. Satoh. On degrees of polynomial interpolations related to elliptic curve cryptography. *Coding and Cryptography*, pages 155–163, 2006.

[75] T. Satoh. On polynomial interpolations related to Verheul homomorphisms. *LMS J. Comput. Math*, 9:135–158, 2006.

[76] T. Satoh. Closed formulae for the Weil pairing inversion. *Finite Fields and Their Applications*, 14(3):743–765, 2008.

[77] O. Schirokauer. Discrete logarithms and local units. *Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences*, 345(1676):409–423, 1993.

[78] M. Scott. Authenticated ID-based key exchange and remote log-in with simple token and PIN number. Technical report, Cryptology ePrint Archive, Report 2002/164, 2002.

[79] M. Scott. Faster pairings using an elliptic curve with an efficient endomorphism. *Progress in Cryptology-INDOCRYPT 2005*, pages 258–269, 2005.

[80] M. Scott. On the efficient implementation of pairing-based protocols. *Cryptography and Coding*, pages 296–308, 2011.

[81] M. Scott and P. Barreto. Generating more MNT elliptic curves. *Designs, Codes and Cryptography*, 38(2):209–217, 2006.

[82] M. Scott, N. Benger, M. Charlemagne, L. Dominguez Perez, and E. Kachisa. Fast hashing to G_2 on pairing-friendly curves. Technical report, Cryptology ePrint Archive, Report 2008/530, 2008.

[83] M. Scott, N. Benger, M. Charlemagne, L. Dominguez Perez, and E. Kachisa. Fast hashing to G_2 on pairing-friendly curves. *Pairing-Based Cryptography–Pairing 2009*, pages 102–113, 2009.

[84] M. Scott, N. Benger, M. Charlemagne, L. Dominguez Perez, and E. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. *Pairing-Based Cryptography–Pairing 2009*, pages 78–88, 2009.

[85] NSA Fact Sheet. Suite B cryptography. http://www.nsa.gov/ia/industry/crypto_suite_b.cfm.

[86] J. Silverman. *The Arithmetic of Elliptic Curves*, volume 106. Springer, 2009.

[87] F. Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010.

[88] E. Verheul. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. *Advances in Cryptology–EUROCRYPT 2001*, pages 195–210, 2001.

[89] J. von zur Gathen. Efficient and optimal exponentiation in finite fields. *Computational Complexity*, 1(4):360–394, 1991.

[90] B. Waters. Efficient identity-based encryption without random oracles. *Advances in Cryptology–EUROCRYPT 2005*, pages 557–557, 2005.

[91] C. Zhao, D. Xie, F. Zhang, J. Zhang, and B. Chen. Computing bilinear pairings on elliptic curves with automorphisms. *Designs, Codes and Cryptography*, 58(1):35–44, 2011.

[92] C. Zhao, F. Zhang, and D. Xie. Reducing the complexity of the Weil pairing computation. Technical report, Cryptology ePrint Archive, Report 2008/212, 2008.