

Pairing Computation on Hyperelliptic Curves of Genus 2

Colm Ó hÉigearthaigh

Bachelor of Science in Computer Applications

A Dissertation submitted in fulfilment of the
requirements for the award of
Doctor of Philosophy (Ph.D.)

to the



Dublin City University

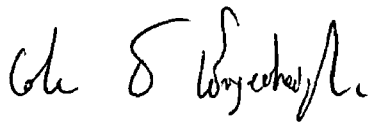
Faculty of Engineering and Computing, School of Computing

Supervisor: Dr. Michael Scott

October, 2006

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work

Signed 

Student ID 99387212

Date October 2006

Contents

Abstract	vi
Acknowledgements	vii
List of Algorithms	viii
List of Tables	ix
1 Introduction	1
1.1 Public Key Cryptography	1
1.2 Bilinear Pairings	5
1.3 Cryptographic Applications of Bilinear Pairings	7
1.3.1 A one-round, three-person key agreement protocol	7
1.3.2 Identity based encryption	8
1.4 Motivation for this Work	9
2 Mathematical Background	13
2.1 Introduction	13
2.2 Finite Fields	14
2.3 Hyperelliptic Curves	18
2.4 Implementing Hyperelliptic Curve Cryptography	27
2.4.1 Elliptic curves	28
2.4.2 Genus 2 curves	31

2.5	The Tate Pairing	34
2.6	Conclusion	42
3	Optimisations to Miller's Algorithm	43
3.1	Introduction	43
3.2	Early Optimisations	44
3.3	Squared Pairings	51
3.4	Pairings on Hyperelliptic Curves	52
3.5	Compressed Pairings	57
3.6	The Weil Pairing	62
3.7	More Recent Optimisations	68
3.8	Conclusion	69
4	Pairings on Supersingular Genus 2 Curves over \mathbb{F}_{2^m}	71
4.1	Introduction	71
4.2	The Curve	72
4.3	Curve Arithmetic	77
4.3.1	Finite field arithmetic	77
4.3.2	Octupling	81
4.3.3	Using degenerate divisors	83
4.3.4	Octupling functions for the Tate pairing	88
4.3.5	The final exponentiation	90
4.4	Computing the Tate Pairing	91
4.4.1	Using an octic basis	91
4.4.2	Precomputing the first point	95
4.4.3	Absorbing powers of 8	99
4.5	Experimental Results	100
4.6	Conclusion	103

5	The η_T Pairing	105
5.1	Introduction	105
5.2	The Theory of the η_T Pairing	106
5.3	The Genus 2 η Pairing	109
5.3.1	Finding a suitable value for T	109
5.3.2	Optimising the arithmetic	112
5.4	Avoiding the Final Exponentiation	113
5.5	The Genus 2 η_T Pairing	118
5.5.1	Finding a suitable value for T	119
5.5.2	Optimising the arithmetic	120
5.6	Experimental Results	123
5.7	Conclusion	128
6	Pairings on Supersingular Genus 2 Curves over \mathbb{F}_p	131
6.1	Introduction	131
6.2	The Curve	132
6.3	Curve Arithmetic	138
6.3.1	Finite field arithmetic	138
6.3.2	Evaluating the line functions	140
6.3.3	The final exponentiation	142
6.4	Computing the Tate Pairing	143
6.4.1	Modifying Miller's algorithm	143
6.4.2	Using denominator elimination	147
6.4.3	Theoretical analysis	149
6.5	Experimental Results	151
6.6	Conclusion	155
7	Conclusion	156
7.1	Review	156

7.2	Open Questions	158
	Bibliography	161
A	Formulae	174
A.1	Absorbing powers of 8 for the genus 2 η pairing	174
A.2	Absorbing powers of 8 for the genus 2 η_T pairing	176
A.3	Unrolling the $\alpha\beta$ multiplication	178

Abstract

Bilinear pairings have been recently used to construct cryptographic schemes with new and novel properties, the most celebrated example being the Identity Based Encryption scheme of Boneh and Franklin. As pairing computation is generally the most computationally intensive part of any pairing-based cryptosystem, it is essential to investigate new ways in which to compute pairings efficiently.

The vast majority of the literature on pairing computation focuses solely on using elliptic curves. In this thesis we investigate pairing computation on supersingular hyperelliptic curves of genus 2. Our aim is to provide a practical alternative to using elliptic curves for pairing based cryptography. Specifically, we illustrate how to implement pairings efficiently using genus 2 curves, and how to attain performance comparable to using elliptic curves.

We show that pairing computation on genus 2 curves over \mathbb{F}_{2^m} can outperform elliptic curves by using a new variant of the Tate pairing, called the η_1 pairing, to compute the fastest pairing implementation in the literature to date. We also show for the first time how the final exponentiation required to compute the Tate pairing can be avoided for certain hyperelliptic curves.

We investigate pairing computation using genus 2 curves over large prime fields, and detail various techniques that lead to an efficient implementation, thus showing that these curves are a viable candidate for practical use.

Acknowledgements

Firstly, I'd like to thank Mike Scott for being a great supervisor. I hope I repaid the faith you showed in me by taking me on as a student 3 years ago.

Thanks also to Steven Galbraith for being in effect my co-supervisor. How you had the patience to answer literally hundreds of emails, often consisting of stupid questions, I'll never know! This thesis would not have happened were it not for your help and encouragement.

I'd also like to thank all of the people that I wrote papers with. Your professionalism and enthusiasm was always inspiring.

Thanks to all of the postgrads in CA, who made the last 3 years a rewarding experience. Rather than list names and risk leaving someone out, I'll just say thanks to everyone who helped me when I needed it.

I'd like to thank everyone in my family for their support, advice and encouragement. Thanks to Aine and Micheal for supporting me in every possible way and for being brilliant role-models. In particular, thanks to Aoileann and Eug for putting up with me for the last 3 years. It would have been a lot harder if it weren't for you.

Finally, to Gemma, whose love and kindness made writing up a lot easier than it would have otherwise been.

List of Algorithms

1	Divisor Composition	22
2	Divisor Reduction	22
3	Miller's algorithm to compute the Tate pairing	41
4	The Duursma-Lee algorithm for the curve $E: y^2 = x^3 - x + d$ over \mathbb{F}_{3^m} , $d = \pm 1$	55
5	Kwon's algorithm for the curves $E: y^2 + y = x^3 + x + d$ over \mathbb{F}_{2^m} , $d = \{0, 1\}$	56
6	Computing Lucas sequence elements	58
7	Doubling of a divisor $[u \ v]$ on the curves C_d	78
8	Octupling of a divisor $[u \ v]$ on the curves C_d	83
9	The genus 2 η pairing	114
10	The genus 2 η_T pairing when $m = 103$	124
11	Miller's algorithm to compute the Tate pairing, as per Galbraith et al [31]	144
12	An improved algorithm to compute the Tate Pairing	146

List of Tables

2 1	The ratios S/M and I/M in MIRACL	17
2 2	The maximum embedding degrees of supersingular hyperelliptic curves over \mathbb{F}_q [96]	27
2 3	Cost of group arithmetic for elliptic curves in \mathbb{F}_p	29
2 4	Cost of group arithmetic for elliptic curves in \mathbb{F}_{2^m}	29
2 5	Cost of group arithmetic for genus 2 curves in \mathbb{F}_p	32
2 6	Cost of group arithmetic for genus 2 curves in \mathbb{F}_{2^m}	33
3 1	Some supersingular elliptic curves with low k	47
3 2	Distortion maps for (most of) the curves given in Table 3 1	48
3 3	Minimum bitlengths of n and q^k	64
3 4	Operation counts for each bit of n	64
4 1	Supersingular genus 2 curves over \mathbb{F}_2	74
4 2	Supersingular genus 2 curves over \mathbb{F}_2 with $k = 12$	74
4 3	\mathbb{F}_{2^m} , where $\#\text{Pic}_C^0(\mathbb{F}_{2^m})$ is equal to a small cofactor times a prime	77
4 4	Experimental results for the curve C_1 over $\mathbb{F}_{2^{79}}$	102
4 5	Experimental results for the curve C_0 over $\mathbb{F}_{2^{103}}$	102
5 1	Experimental results - 950-bit security level	126
5 2	Experimental results - 1230-bit security level	126
6 1	The prime subgroup order n for each security level	135

6 2	Comparison of the cost of doubling in $\text{Pic}_C^0(\mathbb{F}_p)$	136
6 3	Formulae for doubling for the curve $C: y^2 = x^5 + a$	137
6 4	Experimental results for the final exponentiation	143
6 5	Complexity of function calculation per iteration in Miller's Algorithm	147
6 6	Theoretical complexity of Miller's algorithm	151
6 7	Security Parameters	152
6 8	Experimental results - (160/1024) security level	153
6 9	Experimental results - (192/2048) security level	153
6 10	Experimental results - (224/4096) security level	153

Chapter 1

Introduction

1.1 Public Key Cryptography

In 1976, a seminal paper by Diffie and Hellman [22] gave a solution to the key agreement problem, and introduced the revolutionary concept of public key cryptography. Symmetric key cryptography uses a single secret key for both encryption and decryption purposes. In this context, the key agreement problem is to devise an efficient protocol to allow multiple parties to agree upon a secret key over an insecure channel, even if the participants in the protocol have not met before. The security of Diffie and Hellman's elegant solution to this problem is based on the intractability of the so-called Diffie-Hellman Problem in a cyclic Abelian group. However, Diffie and Hellman restrict this definition to the multiplicative group of a finite field, denoted \mathbb{F}_q^* , in their paper.

Aside from the problem of key agreement, symmetric key cryptography has inherent problems in distributing keys in a practical and secure manner, thus hampering its deployment in commercial or digital environments. Public key cryptography consists of two keys, one which is kept public and distributed freely, and the other which is private. The two keys are linked by a one-way function, such that knowledge of the public key reveals no information about the private key. However, a party that knows the private key can decrypt information that is encrypted with the public key. The security of a public key scheme re-

lies on a problem that is believed to be intractable if an adversary lacks certain information, such as the Discrete Logarithm Problem or the Diffie-Hellman Problem mentioned above

In this section, let G_1 be an additively written cyclic group of prime-order n with generator P , such that $G_1 = \langle P \rangle_n$

Definition 1 *The Discrete Logarithm Problem (DLP) in G_1 is the following given $(P, [x]P) \in G_1^2$ find the integer $x \in [0, n - 1]$, where $[x]P$ denotes $\underbrace{P + P + \dots + P}_{x \text{ times}}$*

Pohlig and Hellman [89] showed that an instance of the DLP in an arbitrary cyclic group can be reduced to an instance of the DLP in a prime-order subgroup. Therefore, the order n of G_1 should be a large prime number, or at least divisible by a large prime that is approximately the size of n . The Pollard-rho algorithm [90] is the best algorithm that is known for solving the DLP in a generic group (see also Pollard's Kangaroo method [90]), and it has a fully exponential running time of $\sqrt{\pi n/2}$ group operations. However, Nechaev [87] and Shoup [108] showed that the best possible algorithm to solve the DLP in a generic group runs in time $\Omega(\sqrt{n})$. This shows that Pollard-rho is essentially the best possible generic algorithm to attack the DLP. As Pollard-rho has a fully exponential running time, the DLP is an intractable problem in the abstract setting.

However, every cyclic group of order n is isomorphic to the additive group of integers (modulo n) with generator 1, and the DLP is trivial to solve in this group. This implies that the difficulty of the DLP in a particular group depends on the representation of the group elements. In other words, although the best possible generic algorithm to attack the DLP has an exponential running time, there may be efficient algorithms to attack the DLP in specific groups, that exploit the way that group elements are represented. Therefore, it is crucial to consider attacks on specific groups when choosing a group to implement cryptographic schemes based upon the intractability of the DLP.

The Diffie-Hellman Problem is closely related to the DLP.

Definition 2 *The (computational) Diffie-Hellman Problem (DHP) in G_1 is the following given $(P, [a]P, [b]P) \in G_1^3$ find the element $[ab]P \in G_1$*

The DHP reduces to the DLP in polynomial time, meaning that if the DLP is tractable in a given group, then the DHP is also tractable. To see why this is so, let $(P, [a]P, [b]P)$ be an instance of the DHP in G_1 . Then an adversary can compute the DHP by first finding the integer a by computing the DLP instance $(P, [a]P)$, and then computing $[a]([b]P) = [ab]P$. There is some evidence that the DLP also reduces to the DHP in polynomial time e.g., see den Boer [10], Maurer [76], and Maurer and Wolf [77]. However, this remains unproven for the general case.

To motivate the discussion on the DLP and DHP, we briefly describe the key agreement protocol due to Diffie and Hellman in the case of three parties, which requires two rounds of communication. The order n and the generator P of the group are public parameters. Each party $i \in [0, 1, 2]$ generates a secret integer $x_i \in [0, n - 1]$, and computes the element $[x_i]P$. In the first round of communication, each participant sends their $[x_i]P$ value to one of the other participants, such that each party receives a value. Each participant then computes the multiple of this element by their secret integer x_i , and sends this value again to another participant. Each party then has the shared value $K = [x_0x_1x_2]P$. Any eavesdropper is left with the task of computing K given $(P, [x_0]P, [x_1]P, [x_2]P, [x_0x_1]P, [x_1x_2]P, [x_0x_2]P)$ which implies solving the DHP. When all of the participants have a shared secret, some publicly agreed method is used to extract a key from it, which can then be used in a symmetric key cryptosystem.

The first practical public key encryption and signature scheme was devised by Rivest, Shamir and Adleman (RSA) [93] in 1978. The security of RSA is based on the so-called RSA assumption, a problem believed to be equivalent to the integer factorisation problem. In 1985, the ElGamal [26] cryptosystem was published, which was the first complete cryptographic scheme for encryption/decryption that was based on the intractability of the DLP. At this point, cryptographic schemes that used the DLP as a cryptographic primitive typically followed the original Diffie and Hellman paper, by using the multiplicative group of a suitable finite field \mathbb{F}_q of characteristic p . Although the multiplicative group of a finite field is easy to describe and to implement, the DLP in this setting is vulnerable to sub-exponential

time index calculus attacks. This means that large cryptographic key sizes must be used in practice to maintain security levels.

As the DLP is described in a generic setting, it is natural to examine groups other than \mathbb{F}_q^* . The ideal candidate group would be impervious to all attacks that are faster than the generic Pollard-rho algorithm. The other requirements are that the group elements can be represented in a compact manner, that the group operation can be computed efficiently and that the group order can be computed in polynomial time. Schnorr [100] proposed using a subgroup of prime order n of \mathbb{F}_q^* , where n can be substantially smaller than q . In 1985, Miller [83] and Koblitz [63] independently suggested using the group of rational points on an elliptic curve over a finite field \mathbb{F}_q . No sub-exponential attacks are known to exist for this particular group, and thus key sizes can remain small. The group elements are simply points on the curve, and the group operation corresponds to the inexpensive geometric chord-and-tangent operation. The group order can also be computed efficiently and thus elliptic curves fulfil all of the requirements for use in DLP based cryptosystems.

In 1989, Koblitz [64] suggested using a more general class of curves over \mathbb{F}_q , namely hyperelliptic curves of arbitrary genus. We note that it is not theoretically exact to equate elliptic curves with hyperelliptic curves of genus 1. However, for cryptographic purposes this equivalence holds true, as we concentrate on the arithmetical properties which are the same in both cases, and thus elliptic curves are also automatically considered.

The set of rational points on a hyperelliptic curve of genus $g > 1$ over \mathbb{F}_q , denoted $C(\mathbb{F}_q)$, does not form a group. Instead the divisor class group of degree zero is used, denoted $\text{Pic}_C^0(\mathbb{F}_q)$. The group elements can be represented in a compact manner, and an efficient algorithm due to Cantor [14] exists to perform the group arithmetic. It remains to examine the security of $\text{Pic}_C^0(\mathbb{F}_q)$. The running time for the Pollard-rho algorithm in $\text{Pic}_C^0(\mathbb{F}_q)$ is $O(q^{g/2})$. The best index calculus attack on the DLP in $\text{Pic}_C^0(\mathbb{F}_q)$ is due to Gaudry et al [40]. The complexity of this attack is $O(q^{2-2/g})$, and is therefore faster than Pollard-rho for genus $g \geq 3$, as long as q is sufficiently large. However, hyperelliptic curves of genus 2 are invulnerable to these attacks, and along with elliptic curves are a

good candidate for implementing DLP based cryptosystems

1.2 Bilinear Pairings

Bilinear pairings were first introduced to cryptology by Menezes et al [80] and Frey and Ruck [30], to attack instances of the DLP on elliptic curves and hyperelliptic curves. However, in 2000 Joux [52] and Sakai et al [99] showed how bilinear pairings could be used constructively, to build cryptographic protocols with unique properties. The literature now contains a vast amount of pairing based protocols, many of which provide long-desired solutions to outstanding protocol questions. In this section, bilinear pairings and their associated hard problems are defined. For a more detailed discussion on this topic, the reader is referred to Menezes [79], or Galbraith and Menezes [34].

Let $G_1 = \langle P \rangle_n$ be an additively written Abelian group of prime order n and identity element ∞ , and let G_2 be a multiplicatively written Abelian group of prime order n with identity element 1. A restricted definition of a bilinear pairing is now provided that is suitable for most cryptographic applications. This definition, which is commonly deployed both in theory and in practice, restricts both input elements to the pairing as belonging to the same additive group.

Definition 3 *A bilinear pairing on (G_1, G_2) is a map*

$$e : G_1 \times G_1 \rightarrow G_2$$

that satisfies the following requirements

- 1 (Bilinearity) For all $R, S, T \in G_1$, $e(R+S, T) = e(R, T)e(S, T)$ and $e(R, S+T) = e(R, S)e(R, T)$
- 2 (Non-degeneracy) $e(P, P) \neq 1$, where $P \neq \infty$
- 3 (Computability) e can be efficiently computed i.e. in polynomial time

For all $S, T \in G_1$, a bilinear pairing has the properties

- 1 $e(S, \infty) = e(\infty, S) = 1$
- 2 (Bilinearity) $e([a]S, [b]T) = e(S, T)^{ab}$ for all $a, b \in \mathbb{Z}$
- 3 (Symmetry) $e(S, T) = e(T, S)$

It is the bilinearity property that allows the development of new and exciting cryptographic protocols, as detailed later on in this chapter. A large number of pairing based cryptosystems rely on the property of symmetry associated with the restriction of both input elements to the same additive group. The non-degeneracy requirement ensures that cryptographic applications are not trivial. Finally, the stipulation that the pairing e can be efficiently computed is fulfilled by computing either the Weil or Tate pairings using the degree zero divisor class group of a hyperelliptic curve, as will be detailed in the following chapter.

The bilinearity property implies that the DLP in G_1 can be reduced efficiently to the DLP in G_2 . Let $(P, [x]P)$ be an instance of the DLP in G_1 . Then the bilinearity property gives the equality $e(P, [x]P) = e(P, P)^x \in G_2$. Therefore, solving the DLP instance $(P, [x]P) \in G_1^2$ is equivalent to solving the DLP instance $(e(P, P), e(P, [x]P)) \in G_2^2$.

Definition 4 Let e be a bilinear pairing on (G_1, G_2) . The (computational) Bilinear Diffie-Hellman Problem (BDHP) is the following: given $(P, [a]P, [b]P, [c]P) \in G_1^4$ compute $e(P, P)^{abc} \in G_2$.

The BDHP is assumed to be just as hard as the DHP in G_1 and G_2 . It is known that if the DHP is tractable in either G_1 or G_2 , then the BDHP is also tractable. However, it is not known if the converse is true. If the DHP in G_1 is tractable, then the BDHP can be computed as $[ab]P$ and then $e([ab]P, [c]P) = e(P, P)^{abc}$. Alternatively, if the DHP in G_2 is tractable, then the BDHP can be computed by letting $g = e(P, P)$, and then computing $g^{ab} = e([a]P, [b]P)$, $g^c = e(P, [c]P)$ and g^{abc} . Therefore, the DHP (and hence the DLP) in both G_1 and G_2 needs to be intractable to guarantee the (assumed) security of a pairing based cryptosystem that uses the BDHP as a cryptographic primitive.

Interestingly, Joux and Nguyen [53] show that the Decisional Diffie-Hellman Problem is efficiently computable in G_1 using bilinear pairings, even if the DHP is intractable

Definition 5 *The Decisional Diffie-Hellman Problem (DDHP) is the following given $(P, [a]P, [b]P, [c]P) \in G_1^4$ decide whether $[c]P = [ab]P$*

The DDHP in G_1 can be computed as follows. Let $\gamma_1 = e(P, [c]P) = e(P, P)^c$ and $\gamma_2 = e([a]P, [b]P) = e(P, P)^{ab}$. Then $[c]P = [ab]P$ if and only if $\gamma_1 = \gamma_2$.

1.3 Cryptographic Applications of Bilinear Pairings

As mentioned in the previous section, following the introduction of pairings in a constructive manner, a large amount of attention has been devoted to using bilinear pairings to build cryptosystems with new and novel properties. As motivation for the work in this thesis, brief descriptions of two important pairing based protocols are given, namely the key agreement protocol of Joux and the identity based encryption scheme of Boneh and Franklin.

1.3.1 A one-round, three-person key agreement protocol

As noted previously, the Diffie-Hellman key agreement protocol can be used to agree keys between three participants in only two rounds. However, in 2000 Joux [52] showed the surprising result that it is possible to achieve this in only one round using bilinear pairings, thus solving a long outstanding question as to whether this was possible at all. Here this protocol is described as modified by Verheul [113] to reduce the bandwidth requirements. Again, each party $i \in [0, 1, 2]$ generates a secret integer $x_i \in [0, n - 1]$, but this time broadcasts the element $[x_i]P$ to both of the other parties, and receives the elements $[x_{i-1}]P$ and $[x_{i+1}]P$ (where the subscript of x is considered modulo 3). Note that the three messages are independent of each other, and thus all communication between the participants can be said to occur in a single round. Each party can then establish a shared secret key as $K = e([x_{i-1}]P, [x_{i+1}]P)^{x_i} = e(P, P)^{x_{i-1}x_i x_{i+1}}$. A (passive) eavesdropper must solve an instance of the BDHP to determine the shared key.

Joux's one-round key agreement protocol can be extended to n participants, by using an efficiently computable multilinear map $G_1^{n-1} \rightarrow G_2$, for which a suitable extension of the BDHP is intractable. However, the construction of multilinear maps that can be efficiently computed is an open problem. In fact, Boneh and Silverberg [13] present evidence that it may not be possible to construct such multilinear maps using techniques from algebraic geometry. In any case, Joux's protocol is not feasible from a practical point of view, as it only resists passive attacks, and must have an extra round of communication to resist a man-in-the-middle attack by an active adversary. However, it is useful as an example of how bilinear pairings can be used to provide an elegant solution to cryptographic protocols previously thought impossible.

1.3.2 Identity based encryption

In 1984, Shamir [107] called for a public key, identity based encryption scheme in which the public key can be an arbitrary string. Shamir's original motivation for this scheme was to simplify the management of certificates in email systems. For example, if Alice wants to send an encrypted email to Bob, she encrypts the message using Bob's public key string, which simply corresponds to his email address. This differs considerably from traditional certificate-based schemes, where Alice needs to obtain the certificate containing Bob's public key from some trusted source. When Bob receives the encrypted message he contacts a third party, known as the Private Key Generator (PKG). Bob authenticates himself to the PKG and obtains his private key. Of course, this means that key-escrow is inherent in Identity Based Encryption (IBE), as the PKG knows Bob's private key. By appending a future date to Bob's public key string, Alice can also ensure that a fresh key is used if she thinks that Bob's private key has been compromised.

In 2000, Sakai et al. [99] suggested that bilinear pairings could be used to achieve identity based cryptography. In 2001, Boneh and Franklin [11] realised the first concrete implementation of IBE, which is based on bilinear pairings (and hence on the intractability of the BDHP). Here, the basic idea of Boneh and Franklin's IBE scheme is presented

Let $e: G_1 \times G_1 \rightarrow G_2$ be a bilinear pairing for which the BDHP is intractable, and let $H_1: \{0, 1\}^* \rightarrow G_1 \setminus \{\infty\}$ and $H_2: G_2 \rightarrow \{0, 1\}^l$ be cryptographic hash functions, where l is the bit length of the plaintext m to be encrypted. The PKG selects a private key s at random from $[1, n - 1]$, and computes its public key as $Q = [s]P$. It is assumed that all entities have an authentic copy of Q .

Bob's private key is $d_B = [s]Q_B$, where $Q_B = H_1(ID_B)$, and ID_B is the public key string associated with Bob's identity. Note that computing d_B from (P, Q, Q_B) is an instance of the DHP in G_1 , which only the PKG can compute as it has access to the secret value s . Alice encrypts a message $m \in \{0, 1\}^l$, by first randomly selecting an integer $r \in [1, n - 1]$ and then computing the following values

$$Q_B = H_1(ID_B), C_1 = [r]P, C_2 = m \oplus H_2(e(Q_B, Q)^r)$$

The ciphertext that Alice sends to Bob is then (C_1, C_2) . Bob can recover the original message m from the ciphertext (C_1, C_2) by using his private key d_B to compute

$$m = C_2 \oplus H_2(e(d_B, C_1))$$

To see how the bilinearity property of the pairing enables the decryption of the ciphertext, observe that

$$e(d_B, C_1) = e([s]Q_B, [r]P) = e(Q_B, [s]P)^r = e(Q_B, Q)^r$$

An adversary who attempts to recover m from the ciphertext (C_1, C_2) has to compute $e(Q_B, Q)^r$ from (P, Q_B, Q, C_1) , which is an instance of the BDHP.

1.4 Motivation for this Work

Bilinear pairings have been described thus far in a generic sense. To implement pairing based cryptographic protocols, such as the two protocols described in the previous section,

it must be shown how to construct pairings in a more concrete manner. There are only two bilinear pairings that are of interest for cryptographic purposes, namely the Tate pairing and the Weil pairing. In these cases, the group G_1 is (loosely) the subgroup of order n of the degree zero divisor class group of a hyperelliptic curve defined over a finite field \mathbb{F}_{q^k} , where k is known as the embedding degree of the curve, and the group G_2 is the group of the n th roots of unity in \mathbb{F}_{q^k} . Ideally, the embedding degree should be large enough to protect against index calculus attacks in G_2 , yet small enough to allow for the efficient implementation of \mathbb{F}_{q^k} . In an unpublished manuscript in 1986 (later published in 2004), Miller [82, 84] described how to implement the Weil pairing efficiently using a double-and-add algorithm. This algorithm, now known as Miller's algorithm, can be easily adapted to compute the Tate pairing.

Computing either the Weil or the Tate pairing in an efficient manner is essential, as pairing computation is generally the most intensive task in any pairing based cryptosystem. Therefore, it is important to investigate ways in which to speed up pairing computation, if cryptographic protocols that are based on pairings are to be adopted in practice. In recent years, a large body of work has appeared in the literature to address this issue, and much progress has been made in implementing bilinear pairings in an efficient manner as a result. Indeed, over the course of twenty years, the time to compute cryptographically secure bilinear pairings has decreased dramatically, from several minutes to only a few milliseconds [5, 103]. Although the performance of pairing-based cryptosystems is approaching that of cryptographic schemes such as RSA, there is still considerable motivation to improve pairing computation, as it is an open question as to whether pairing based schemes might offer improved performance over traditional public key protocols.

The vast majority of the literature on pairing computation focuses solely on using elliptic curves. The reasons for using elliptic curves to compute pairings are largely the same reasons as to why elliptic curves are preferred for discrete-log based cryptosystems. Firstly, the description of the group elements as rational points on the curve is far simpler than the complicated divisor theory involved in using hyperelliptic curves of genus greater than one.

Secondly, the representation of group elements in the elliptic curve case requires only 2 field elements, as opposed to 4 in the genus 2 case, 6 in the genus 3 case, etc. Thirdly, the group law is vastly simplified in the elliptic case as it corresponds to the simple manipulation of geometric lines, as opposed to using Cantor's algorithm for composition and reduction of divisors. The group operation for elliptic curves costs less in general than the group law for hyperelliptic curves of higher genus.

However, despite the numerous apparent advantages of using elliptic curves, there are compelling reasons to investigate pairing computation on other hyperelliptic curves. Firstly, no efficient pairing implementations on curves of genus $g > 1$ exist, meaning that protocol designers can only consider using elliptic curves. It is of considerable theoretical and practical importance to provide an alternative to using elliptic curves for pairing computation. However, we focus solely on using hyperelliptic curves of genus 2. The added complexity of the group law for hyperelliptic curves of genus $g > 2$ makes the group arithmetic difficult to implement in an efficient manner, and there are very few examples of such curves that are useful for pairing based cryptography.

Secondly, considerable effort has gone into deriving explicit formulae for the group law for genus 2 curves, see Lange [70], which improve substantially on the generic algorithm due to Cantor [14]. Avanzi [1] uses these formulae to show that a careful implementation of scalar multiplication on genus 2 curves over large prime fields is extremely competitive with elliptic curves. It is natural to wonder then whether pairing computation on genus 2 curves is competitive also with the equivalent elliptic curve implementations. Thirdly, curves of genus 2 have a richer algebraic structure than those of genus 1. It is possible that this additional structure can be exploited in some way to speed up the computation.

Rubin and Silverberg [96] show that the maximum embedding degree k of supersingular genus 2 curves is 12 over \mathbb{F}_{2^m} and 6 over \mathbb{F}_p . The interesting value for security purposes is k/g , where g is the genus of the curve. The maximum security parameter for genus 2 curves is attained in characteristic 2 ($k/g = 6$), as opposed to characteristic 3 for elliptic curves ($k/g = 6$). This is another reason to consider using genus 2 curves, as working in

characteristic 2 is preferable to using fields of characteristic 3. Supersingular genus 2 curves over \mathbb{F}_{2^m} with a maximum embedding degree of $k = 12$ are known to exist. However, there are no known curves over \mathbb{F}_p with an embedding degree of $k = 6$. Instead, a supersingular genus 2 curve defined over \mathbb{F}_p with an embedding degree of $k = 4$ is known to exist. The vast majority of efficient implementations of finite field arithmetic use either binary extension fields \mathbb{F}_{2^m} , or large prime fields \mathbb{F}_p . Coincidentally, supersingular genus 2 curves only have interesting embedding degrees over these fields.

In this thesis, pairing computation on supersingular hyperelliptic curves of genus 2 over both \mathbb{F}_{2^m} and \mathbb{F}_p is investigated, using curves with the maximum embedding degree that is known in each case. Specifically, it is illustrated how to implement pairings efficiently using these curves, and how to attain performance comparable to using elliptic curves. The handful of papers that exist on this topic in the literature yield inefficient implementations relative to elliptic curves. In this thesis, the open question as to whether genus 2 curves provide a viable alternative to using elliptic curves for pairing computation is answered in the affirmative. This thesis deals solely with improvements to the actual computation of pairings, and does not focus at all on cryptographic protocols that are based on bilinear pairings.

The structure of this thesis is as follows. Chapter 2 provides an overview of elliptic and hyperelliptic curve cryptography, the Weil and Tate pairings, and Miller's algorithm. Chapter 3 is a literature review of papers dealing with advances in the computation of pairings. The research contribution of this thesis is split into three chapters. Chapter 4 explores the computation of the Tate pairing on a supersingular genus 2 curve over \mathbb{F}_{2^m} . Chapter 5 uses a new variant of the Tate pairing, called the η_T pairing, to compute the fastest pairing implementation in the literature. It is also shown for the first time how the final exponentiation required to compute the Tate pairing can be avoided for certain supersingular curves. Chapter 6 details the computation of pairings on a supersingular genus 2 curve over \mathbb{F}_p , vastly improving on results available in the literature. A new variant of Miller's algorithm is also described. Finally, the thesis is concluded in chapter 7.

Chapter 2

Mathematical Background

2.1 Introduction

Cryptography that is based on the properties of algebraic curves is not an easy discipline as it involves deep mathematical concepts. In this chapter, an overview of the mathematical techniques that are fundamental to the work in this thesis is provided. This chapter states some of the results given in the previous chapter in a more concrete manner. It is not our intention to be comprehensive, instead the interested reader can pursue any of the numerous references that are cited throughout the chapter.

Firstly, finite fields are examined. Finite fields play an important role in modern cryptography, and it is essential to represent finite field elements in such a way that allows for an efficient implementation. Various bases for representing finite fields, as well as details about how to implement arithmetic, are investigated. The relative cost of various fundamental finite field operations is also explored. See Lidl and Niederreiter [75] for a more comprehensive treatment on this topic.

Secondly, hyperelliptic curves and their applications in cryptography are examined. Some divisor theory is given, and a group is constructed with a compact representation of the group elements. An algorithm is given to perform the group operation. It is then detailed how this group is suitable for cryptography, and the DLP is defined in this context.

Menezes et al [81] provide a good “elementary” overview of using hyperelliptic curves in cryptography. Another thorough review is found in Jacobson et al [56], Galbraith and Menezes [34], and in various chapters of Cohen et al [19]. A compact treatment of this topic is given in Hietalahti [49].

Thirdly, two concrete implementations of hyperelliptic curves are examined, namely hyperelliptic curves of genus 1 (elliptic curves), and hyperelliptic curves of genus 2. The vast majority of implementations in the literature use one of these types of curves. Fourthly, we make concrete the abstract notion of a bilinear pairing, by introducing the Tate pairing and the Weil pairing. These pairings are given in the more general setting of hyperelliptic curves, rather than using elliptic curves, as is common in the literature. An algorithm is given to compute pairings, and finally the chapter is concluded. Good references on this topic are chapter 9 of Blake et al [9], as well as chapters 6 and 16 of Cohen et al [19].

2.2 Finite Fields

A field is a commutative ring for which every non-zero element has a multiplicative inverse. Let K and L be fields such that $K \subseteq L$. An element $\alpha \in L$ is said to be algebraic over K if there is a polynomial $f(x)$ in one variable with coefficients in K , such that $f(\alpha) = 0$. The field L is an algebraic extension of K if every element of L is algebraic over K .

Definition 6 *An algebraic closure of a field K , is a field \overline{K} containing K , such that \overline{K} is algebraic over K and every nonconstant polynomial with coefficients in \overline{K} has a root in \overline{K} . i.e. \overline{K} is algebraically closed.*

The field K has prime characteristic p , if there is a prime p such that $1 + 1 + \dots + 1 = 0$ (p times), where 1 is the multiplicative identity, and 0 is the additive identity of the field. Otherwise, K is said to have characteristic 0. If a field K has prime characteristic, then K contains the finite field of integers modulo p , i.e. $\{0, 1, \dots, p-1\}$, denoted \mathbb{F}_p . For any prime p , and any positive integer m , there exists a finite field with $q = p^m$ elements. This field is unique up to isomorphism and is denoted \mathbb{F}_q . The algebraic closure of \mathbb{F}_q is defined

as

$$\overline{\mathbb{F}}_q = \bigcup_{i=1}^{\infty} \mathbb{F}_{q^i}$$

The multiplicative group of nonzero elements of \mathbb{F}_q , denoted \mathbb{F}_q^* , forms a cyclic Abelian group of order $q - 1$. For any element $\alpha \in \mathbb{F}_q^*$, then $\alpha^{q-1} = 1$ due to the theorem of Lagrange, and therefore $\alpha^q = \alpha$. This map can be generalised, as in the following definition

Definition 7 The q -th power Frobenius automorphism ϕ_q of $\overline{\mathbb{F}}_q$ is defined as

$$\phi_q \begin{cases} \overline{\mathbb{F}}_q & \rightarrow \overline{\mathbb{F}}_q \\ x & \mapsto x^q \end{cases}$$

for all $x \in \overline{\mathbb{F}}_q$. We will sometimes refer to the p -th power Frobenius automorphism ϕ_p of a finite field \mathbb{F}_q of characteristic p which is defined as

$$\phi_p \begin{cases} \mathbb{F}_q & \rightarrow \mathbb{F}_q \\ x & \mapsto x^p \end{cases}$$

for all $x \in \mathbb{F}_q$

Let \mathbb{F}_q and \mathbb{F}_{q^k} be finite fields, such that \mathbb{F}_{q^k} is a finite extension of \mathbb{F}_q . Then \mathbb{F}_{q^k} can be regarded as a vector space of dimension k over \mathbb{F}_q . This means that there is a basis $(\beta_0, \beta_1, \dots, \beta_{k-1})$, where $\beta_i \in \mathbb{F}_{q^k}$, such that every element $\alpha \in \mathbb{F}_{q^k}$ has a unique representation of the form

$$\alpha = \sum_{i=0}^{k-1} a_i \beta_i,$$

where $a_i \in \mathbb{F}_q$. The element $\alpha \in \mathbb{F}_{q^k}$ is denoted by the \mathbb{F}_q -vector $(a_0, a_1, \dots, a_{k-1})$. The addition of two finite field elements is performed on the a_i components of the two elements. However, the multiplication of two elements requires knowledge about the dependencies between the elements of the basis. There are many different bases of \mathbb{F}_{q^k} over \mathbb{F}_q , however

two bases are mainly used in practice

The most commonly implemented basis is the polynomial basis. Let $\mathbb{F}_q[x]$ be the ring of polynomials in x . A polynomial $f \in \mathbb{F}_q[x]$ is said to be irreducible if f has positive degree, and the equation $f = bc$ implies that either $b \in \mathbb{F}_q[x]$ or $c \in \mathbb{F}_q[x]$ is a constant polynomial. If $f \in \mathbb{F}_q[x]$ is an irreducible polynomial over \mathbb{F}_q of degree k , then a finite field with q^k elements is constructed by adjoining a root of f to \mathbb{F}_q . For every finite field \mathbb{F}_q , and every positive integer k , there exists an irreducible polynomial in $\mathbb{F}_q[x]$ of degree k . If $\beta \in \mathbb{F}_{q^k}$ is a root of f , then $(1, \beta, \beta^2, \dots, \beta^{k-1})$ is called the polynomial basis of \mathbb{F}_{q^k} over \mathbb{F}_q . Addition, subtraction and multiplication are performed modulo f . Inversion can be computed using the extended Euclidean algorithm in $\mathbb{F}_q[x]$. Irreducible binomials, trinomials and pentanomials are commonly used to define extensions of a finite field, as they allow for a fast reduction.

A less commonly used basis is known as the normal basis. The element $\beta \in \mathbb{F}_{q^k}$ is said to be normal over \mathbb{F}_q if the elements $(\beta, \beta^q, \beta^{q^2}, \dots, \beta^{q^{k-1}})$ are linearly independent over \mathbb{F}_q . Then the basis $(\beta, \beta^q, \beta^{q^2}, \dots, \beta^{q^{k-1}})$ is called a normal basis of \mathbb{F}_{q^k} over \mathbb{F}_q . For every finite field \mathbb{F}_q and positive integer k , there exists a normal basis of \mathbb{F}_{q^k} over \mathbb{F}_q . Using a normal basis, exponentiating an element $\alpha \in \mathbb{F}_{q^k}$ to the power of q can be achieved with a simple shift of the vector representation. This is particularly useful for finite fields of characteristic two, as squarings become trivial as a result. However, squaring can also be speedily implemented in characteristic 2 using a polynomial basis, by inserting the 0 bit between every other bit of the binary representation of the element, and then reducing modulo the irreducible polynomial. A further advantage of using a polynomial basis is that the multiplication of two elements in the normal basis is complicated and requires the precomputation of a table. Therefore, in this thesis only polynomial bases are considered.

Two particular types of finite fields are commonly used in cryptography, namely binary fields \mathbb{F}_{2^m} , where m is prime to avoid Weil descent attacks (see Gaudry et al. [39]), and large prime fields \mathbb{F}_p . Addition in \mathbb{F}_{2^m} reduces to a bitwise XOR operation, and hence this field is commonly deployed in hardware implementations. \mathbb{F}_{2^m} also has the advantage that

Table 2.1 The ratios S/M and I/M in MIRACL

Finite Field	S/M	I/M
\mathbb{F}_p	0.8 → 1.0	10 → 40
\mathbb{F}_{2^m}	0.1 → 0.25	9 → 13

squaring in this field is substantially faster than multiplication. Note that no irreducible binomials exist over \mathbb{F}_2 , and therefore either a trinomial or pentanomial should be used as the irreducible polynomial. Large prime fields \mathbb{F}_p have the advantage of being simple to implement, as no field extensions are involved. However, the ratio between squarings and multiplications is far larger than for \mathbb{F}_{2^m} . Optimal extension fields \mathbb{F}_{p^d} , where the irreducible polynomial defining the extension allows for a fast reduction, are sometimes also used (particularly in embedded applications). These fields are deployed if it is necessary to avoid the specific disadvantages of \mathbb{F}_{2^m} or \mathbb{F}_p .

Computing the ratios S/M and I/M in \mathbb{F}_p and \mathbb{F}_{2^m} is of considerable interest when implementing finite field arithmetic, where S, M, I denote a squaring, multiplication and inversion respectively. However, the ratios depend on a wide range of parameters, such as the representation used, whether the implementation is in software or hardware, how much optimisation is used, etc. This implies that different implementations yield widely differing estimates. Nonetheless, it is useful to provide these ratios to assess the cost of inversion in particular. The MIRACL [102] library provides a suite of test programs for calculating the ratios S/M and I/M in both \mathbb{F}_{2^m} and \mathbb{F}_p . The implementation of both \mathbb{F}_{2^m} and \mathbb{F}_p includes numerous platform-specific enhancements and is highly optimised. The testing includes different parameters for m and p , that range from 103 to 2048 bits. The results are included in Table 2.1, on our platform of a Pentium IV, 2.8 GHz. Note that the ratio I/M can be far more expensive for \mathbb{F}_p than it is for \mathbb{F}_{2^m} . We emphasise that I/M is expensive in \mathbb{F}_p due mainly to the highly efficient way multiplication is implemented in MIRACL.

If $f \in \mathbb{F}_q[x]$ is an irreducible polynomial of degree k , then f has a root α in \mathbb{F}_{q^k} . All of the roots of f are given by the k distinct elements $(\alpha, \alpha^q, \dots, \alpha^{q^{k-1}}) \in \mathbb{F}_{q^k}$, which are called the conjugates of α with respect to \mathbb{F}_q .

Definition 8 Let $\alpha \in \mathbb{F}_{q^k}$. Then the trace of α is given by

$$\text{Tr}_{\mathbb{F}_{q^k}/\mathbb{F}_q}(\alpha) = \sum_{i=1}^k \alpha^{q^i},$$

and the norm of α is given by

$$\text{N}_{\mathbb{F}_{q^k}/\mathbb{F}_q}(\alpha) = \prod_{i=1}^k \alpha^{q^i}$$

2.3 Hyperelliptic Curves

Definition 9 An (imaginary quadratic) hyperelliptic curve C of genus $g \geq 1$ over the field K is defined by an equation of the form

$$C: y^2 + h(x)y = f(x),$$

where $h(x) \in K[x]$ is of degree at most g and $f(x) \in K[x]$ is monic of degree $2g + 1$

C must be non-singular, meaning that there are no pairs $(x, y) \in \overline{K} \times \overline{K}$ which satisfy both the equation of the curve C , and the partial derivative equations

$$2y + h(x) = 0, \quad h'(x)y - f'(x) = 0$$

When the characteristic of K is not equal to 2, then the equation C can be transformed into $y^2 = f(x)$, where $f(x)$ has degree $2g + 1$, by the change of variables

$$x \rightarrow x, \quad y \rightarrow (y - h(x))/2$$

In this case, the condition on the partial derivatives is met if and only if $f(x)$ has no repeated roots in \overline{K}

Definition 10 Let L be a field containing K . Then the set of L -rational points on the curve C , denoted $L(C)$ is defined to be the set of finite points $\{(x, y) \in L \times L\}$ that satisfy the

equation of the curve C along with the point at infinity ∞ . The set of \overline{K} -rational points $C(\overline{K})$ is denoted C for short.

The opposite of a point $P = (x, y) \in C$, denoted $-P$, is the unique other point on the curve with the same x -coordinate, and is computed as $-P = (x, -y - h(x))$. If $P = \infty$, then $-P = \infty$. If a finite point is equal to its opposite it is called special, otherwise it is said to be ordinary.

Definition 11 A divisor D is a finite formal sum of points on C such that

$$D = \sum m_i(P_i)$$

where $m_i \in \mathbb{Z}$, and $m_i = 0$ for all but finitely many $P_i \in C$.

The degree of a divisor is the integer $\sum m_i$. The support of a divisor is the finite set $\{P_i \in C \mid m_i \neq 0\}$. The set of divisors forms a free Abelian group, denoted Div_C , under the addition law

$$\sum n_i(P_i) + \sum m_i(P_i) = \sum (n_i + m_i)(P_i)$$

The (sub)group of divisors of degree 0 is denoted Div_C^0 . The greatest common divisor of two divisors $D_1 = \sum n_i(P_i) \in \text{Div}_C^0$ and $D_2 = \sum m_i(P_i) \in \text{Div}_C^0$ is also an element of Div_C^0 , and is computed as

$$\gcd(D_1, D_2) = \sum \min(m_i, n_i)(P_i) - \left(\sum \min(m_i, n_i)\right)(\infty)$$

Given a point $P \in C$, and a function f considered on C , if $f(P) = 0$ then f is said to have a zero at P . If f is not defined at P , then f is said to have a pole at P , in which case $f(P) = \infty$. The order of a function at a point, or the number of zeros or poles at a point, can be computed in the following way.

Definition 12 Let $G(x, y)$ be a polynomial with coefficients in \overline{K} considered as a function

on C . As y^2 can be repeatedly replaced with $f(x) - h(x)y$, an equivalent polynomial $\overline{G}(x, y)$ can be obtained such that $\overline{G}(x, y) = a(x) - b(x)y$. Then the order of $G(x, y)$ at a point $P \in C$ denoted $\text{ord}_P G$ is computed as follows

1. $P = (x_P, y_P)$ is a finite point. Let \overline{G} be in the form $(x - x_P)^r (a_0(x) - b_0(x)y)$, where $(x - x_P)$ does not divide both $a_0(x)$ and $b_0(x)$. If $a_0(x_P) - b_0(x_P)y_P \neq 0$ then let $s = 0$; otherwise let s be the exponent of the highest power of $(x - x_P)$ which divides $a_0^2(x) + h(x)a_0(x)b_0(x) - f(x)b_0^2(x)$. If P is an ordinary point then define $\text{ord}_P G = r + s$; otherwise define $\text{ord}_P G = 2r + s$.

2. $P = \infty$. Then $\text{ord}_\infty G = -\max(2 \deg(a), 2g + 1 + 2 \deg(b))$.

To any $G(x, y)$ such that $\overline{G} \neq 0$, the divisor $(G) = \sum (\text{ord}_{P_i} G)(P_i)$ is associated. A rational function R on C is defined as a ratio $R = G(x, y)/H(x, y)$, with $\overline{H} \neq 0$. To such a rational function the divisor $(G/H) = (G) - (H) \in \text{Div}_C^0$ is associated. The order of R at a point $P \in C$ is defined to be $\text{ord}_P R = \text{ord}_P G - \text{ord}_P H$, if $H(P) \neq 0$. Note that $\text{ord}_P R$ does not depend on the choice of G and H . Evaluating a divisor $D = \sum m_i(P_i)$ at a rational function R is computed as

$$R(D) = \prod R(P_i)^{m_i},$$

assuming that (R) and D have disjoint support. If R and D are both defined over K , then $R(D) \in K$. The divisor of a rational function is called a principal divisor, and such divisors form a subgroup of Div_C^0 . A principal divisor also has the property that $\sum m_i P_i = \infty$.

Definition 13 The (degree zero) divisor class group $\text{Pic}_C^0(K)$ of C over K is the quotient group of the degree zero divisors defined over K modulo the principal divisors defined over K . It is also known as the Picard group of C .

Two divisors D_1 and D_2 are equivalent (when considered as elements of $\text{Pic}_C^0(K)$), denoted $D_1 \sim D_2$, if their difference $D_1 - D_2$ is a principal divisor, i.e. $D_1 = D_2 + (R)$, for some rational function R .

Definition 14 A semi-reduced divisor D is of the form $D = \sum m_i(P_i) - (\sum m_i)(\infty)$ such that

- 1 All $m_i \geq 0$ and the P_i are finite points
- 2 If $P_i \neq -P_i$ then only one of them occurs in the sum with $m_i \neq 0$
- 3 If $P_i = -P_i$ then $m_i \leq 1$

Any $D \in \text{Div}_C^0$ can be modified by a principal divisor to obtain an equivalent $D_1 \sim D$ such that D_1 is a semi-reduced divisor. This implies that every coset of $\text{Pic}_C^0(K)$ can be represented by a semi-reduced divisor.

The field K is defined from now on to be \mathbb{F}_q , the finite field of q elements. A divisor D is said to be defined over \mathbb{F}_q if $D^\sigma = \sum m_i(P_i^\sigma) = D$, for all automorphisms σ of $\overline{\mathbb{F}_q}$ over \mathbb{F}_q , where $P^\sigma = (\sigma(x), \sigma(y))$ if $P = (x, y)$ and $\infty^\sigma = \infty$. The theorem of Riemann-Roch implies that every element of $\text{Pic}_C^0(\mathbb{F}_q)$ can be represented uniquely by a semi-reduced divisor $D = \sum m_i(P_i) - (\sum m_i)(\infty)$, with the additional property that $\sum m_i \leq g$, where g is the genus of the curve. Semi-reduced divisors with this property are known as reduced divisors.

The degree zero divisor class group $\text{Pic}_C^0(K)$ is isomorphic to the Jacobian of the curve C defined over K , denoted $J_C(K)$. The Jacobian is an Abelian variety of dimension g . $\text{Pic}_C^0(K)$ is also isomorphic to the ideal class group of the function field $K(C)$, which is the field of rational functions on C . Mumford [86] introduced a way of representing a semi-reduced divisor as the greatest common divisor of two principal divisors of functions of the form $u(x)$ and $v(x) - y$. This uses the ideal class group representation, and is extremely useful for implementation.

Definition 15 A (non-trivial) semi-reduced divisor $D = \sum m_i(P_i) - (\sum m_i)(\infty) \in \text{Div}_C^0(K)$ can be represented by two polynomials $[u, v]$ with coefficients in \mathbb{F}_q such that $u(x) = \prod (x - x_i)^{m_i}$ and $v(x_i) = y_i$, with the following properties

- 1 u is monic

$$2 \quad \deg v < \deg u \leq g$$

$$3 \quad u \mid v^2 + vh - f$$

The identity element of $\text{Pic}_C^0(K)$ is represented in Mumford notation as $[1, 0]$. A divisor in Mumford representation $[u, v]$ is reduced if and only if $\deg(u) \leq g$. If D is defined over \mathbb{F}_q , then u and v are also defined over \mathbb{F}_q . However, this does not necessarily mean that the points in the support of the divisor are defined over \mathbb{F}_q .

Cantor [14] showed how to perform group arithmetic in $\text{Pic}_C^0(K)$ using Mumford representation, assuming that $h(x) = 0$ and $p \neq 2$. This algorithm was later generalised by Koblitz [64] to remove these conditions. The group operation is split into two steps. The first step, called composition, takes as input two semi-reduced divisors D_1 and D_2 , and outputs a semi-reduced divisor $D' \sim D_1 + D_2$. This is given in Algorithm 1. The second step of the group operation is called reduction, and reduces the semi-reduced divisor D' to an equivalent reduced divisor. This is given in Algorithm 2.

Algorithm 1 Divisor Composition

INPUT $D_1 = [u_1, v_1], D_2 = [u_2, v_2]$

OUTPUT $D' \sim D_1 + D_2, D' = [u, v]$

- 1 Compute $d_1 = \gcd(u_1, u_2) = e_1u_1 + e_2u_2$
 - 2 Compute $d = \gcd(d_1, v_1 + v_2 + h) = c_1d_1 + c_2(v_1 + v_2 + h)$
 - 3 $s_1 \leftarrow c_1e_1, s_2 \leftarrow c_1e_2, s_3 \leftarrow c_2$
 - 4 $u \leftarrow (u_1u_2)/(d^2)$
 - 5 $v \leftarrow (s_1u_1v_2 + s_2u_2v_1 + s_3(v_1v_2 + f))/d \pmod u$
 - 6 **Return** $[u, v]$
-

Algorithm 2 Divisor Reduction

INPUT $D = [u, v]$ semi-reduced

OUTPUT $D' = [u', v']$ reduced with $D' \sim D$

- 1 $u' \leftarrow (f - vh - v^2)/u, v' \leftarrow (-h - v) \pmod u'$
 - 2 **if** $\deg u' > g$ **then**
 - 3 $u \leftarrow u', v \leftarrow v'$
 - 4 ▷ Go to step 1
 - 5 **end if**
 - 6 ▷ Make u' monic
 - 7 **Return** $[u', v']$
-

$\text{Pic}_C^0(\mathbb{F}_q)$ is a finite Abelian group which fulfills all the criteria given in Chapter 1 for implementing a DLP-based cryptosystem. The group elements can be represented in a compact and simple manner due to the representation of Mumford. The group operation can be performed efficiently due to the algorithm of Cantor. The DLP is defined in the hyperelliptic context as follows

Definition 16 *The Hyperelliptic Discrete Logarithm Problem (HCDLP) in $\text{Pic}_C^0(\mathbb{F}_q)$ is the following: given $(D, [x]D) \in \text{Pic}_C^0(\mathbb{F}_q)^2$, find an integer $x \in [0, \#\text{Pic}_C^0(\mathbb{F}_q) - 1]$ where $[x]D$ denotes $\underbrace{D + D + \dots + D}_{x \text{ times}}$*

Computing $[x]D$ efficiently is essential for cryptosystems based on the intractability of the DLP. It is inefficient to repeatedly add D to itself when x is a large integer. Instead, it is far more efficient to use the double-and-add algorithm (also known as square-and-multiply when using multiplicative notation). The double-and-add algorithm takes $2\lceil \log_2(x) \rceil$ operations in the worst case, and $3\lceil \log_2(x) \rceil / 2$ operations on average, assuming that a doubling is computationally equivalent to an addition. On average, x will have a Hamming weight of $\log_2(x)/2$, which necessitates $\log_2(x)/2$ additions. It is possible to improve the performance of the double-and-add algorithm by using a windowing algorithm to reduce the number of additions. If x is written in non-adjacent form (NAF), it is possible to reduce the number of additions to $\log_2(x)/3$. This method is effective if x does not have a low Hamming weight.

The best attack on the HCDLP is due to Gaudry et al. [40], which has complexity $O(q^{2-2/g})$. This is faster than the generic Pollard-rho attack for genus $g \geq 3$. Therefore, hyperelliptic curves of genus 1 or 2 are typically used when implementing DLP-based cryptography. It remains to examine how to determine the cardinality of the group.

Definition 17 *The Hasse-Weil bound gives a bound on the group order that depends only on the underlying finite field \mathbb{F}_q and the genus g of the curve,*

$$(\sqrt{q} - 1)^{2g} \leq \#\text{Pic}_C^0(\mathbb{F}_q) \leq (\sqrt{q} + 1)^{2g}$$

Therefore, the group order for an arbitrary hyperelliptic curve of genus g over \mathbb{F}_q is roughly given as $\#\text{Pic}_C^0(\mathbb{F}_q) \approx q^g$

Definition 18 The q -th power Frobenius endomorphism ϕ_q on a hyperelliptic curve C defined over \mathbb{F}_q is given by

$$\phi_q \begin{cases} \text{Pic}_C^0(\overline{\mathbb{F}_q}) & \rightarrow \text{Pic}_C^0(\overline{\mathbb{F}_q}) \\ \sum_P m_i(P_i) & \mapsto \sum_P m_i(P_i^{\phi_q}) \end{cases}$$

where $P^{\phi_q} = (x^q, y^q)$ and $\infty^{\phi_q} = \infty$

When a divisor is written in Mumford representation $[u \ v]$, then the q -th power Frobenius endomorphism of $\text{Pic}_C^0(\mathbb{F}_q)$ operates on the coefficients of u and v as the q -th power field automorphism. Therefore, applying the q -th power Frobenius endomorphism to a divisor in Mumford representation requires at most $2g$ operations in \mathbb{F}_q . For a hyperelliptic curve C of genus g defined over \mathbb{F}_q , the Frobenius endomorphism ϕ_q satisfies a characteristic polynomial of degree $2g$ of the form

$$\chi_C(T) = T^{2g} + a_1 T^{2g-1} + \dots + a_g T^g + \dots + a_1 q^{g-1} T + q^g,$$

where $a_i \in \mathbb{Z}$. The characteristic polynomial of the Frobenius endomorphism factors as $\chi_C(T) = \prod_{i=1}^{2g} (T - \alpha_i)$, where the α_i are complex numbers of absolute value \sqrt{q} . Once the values α_i are known, then both the number of the points on the curve C and the cardinality of $\text{Pic}_C^0(\mathbb{F}_{q^r})$, for some integer $r \geq 1$, can be computed efficiently.

Lemma 1 Let C be a hyperelliptic curve of genus g over \mathbb{F}_q and let $\chi_C(T) = \prod_{i=1}^{2g} (T - \alpha_i)$ be the characteristic polynomial of the Frobenius endomorphism. Then for any integer $r \geq 1$

$$\#C(\mathbb{F}_{q^r}) = q^r + 1 - \sum_{i=1}^{2g} \alpha_i^r,$$

and

$$\#\text{Pic}_C^0(\mathbb{F}_{q^r}) = \prod_{i=1}^{2g} (1 - \alpha_i^r)$$

Therefore, it is possible to compute $\#\text{Pic}_C^0(\mathbb{F}_{q^r})$ by first deriving the a_i coefficients of $\chi_C(T)$, and then factoring $\chi_C(T)$ to obtain the α_i . Each a_i coefficient can be obtained by computing $\#C(\mathbb{F}_{q^i})$. For a curve of genus g , this involves computing the number of points on the curve $C(\mathbb{F}_{q^i})$ for $1 \leq i \leq g$. This technique is useful for Koblitz curves, which are curves defined over a small field, and then considered over a large extension field. For example, all of the genus 2 curves in characteristic 2 that are examined in this thesis are defined over \mathbb{F}_2 , but considered over \mathbb{F}_{2^m} . However, this method is not generally useful for determining the cardinality of $\text{Pic}_C^0(\mathbb{F}_p)$ for large p , as computing $\#C(\mathbb{F}_{p^i})$ is a non-trivial task. In this case, an algorithm can be used to compute χ_C directly (i.e. without computing the number of points on the curve). Once χ_C is known, the group order can be computed as $\#\text{Pic}_C^0(\mathbb{F}_p) = \chi_C(1)$. Chapter 17 of Cohen et al [19] provides an overview of such algorithms.

Menezes et al [80] show how to reduce the (HC)DLP in $\text{Pic}_C^0(\mathbb{F}_q)$ to the DLP in $\mathbb{F}_{q^k}^*$ in probabilistic polynomial time using the Weil pairing (actually their paper gives this result for elliptic curves). Frey and Ruck [30] use the Tate pairing to achieve the same effect for curves of arbitrary genus, and hence the attack is frequently referred to by the initials of the authors as MOV/FR. As index calculus attacks exist with sub-exponential complexity in $\mathbb{F}_{q^k}^*$, then the reduction implies that the DLP in $\text{Pic}_C^0(\mathbb{F}_q)$ can be compromised if k is small. k is a positive integer that is defined in the following way.

Definition 19 *Let C be a hyperelliptic curve of genus g over \mathbb{F}_q and let $D \in \text{Pic}_C^0(\mathbb{F}_q)$ be a divisor of prime-order n , which is co-prime to q . Then the embedding degree k of $\langle D \rangle_n$ is the smallest positive integer such that $n \mid q^k - 1$. In other words \mathbb{F}_{q^k} is the smallest field that contains the group of the n th roots of unity.*

When selecting a curve to implement a DLP-based cryptosystem, it is desirable to have

an embedding degree k that is as large as possible to avoid the attack of MOV/FR. For the majority of hyperelliptic curves this is automatically satisfied, and when considering the same curve over different fields, k varies over the whole range $1 \leq k \leq q^g$. However, for a certain class of curves that are called supersingular, the embedding degree k is relatively small, which makes supersingular curves unsuited to implementing DLP-based cryptography. Yet it is precisely this property that makes supersingular curves excellent candidates for pairing based cryptography. An Abelian variety over \mathbb{F}_q is supersingular if it is isogenous (over $\overline{\mathbb{F}_q}$) to a product of supersingular elliptic curves. A hyperelliptic curve C over \mathbb{F}_q is called supersingular if the Jacobian, $J_C(\mathbb{F}_q)$, is supersingular.

As hyperelliptic curves of genus g over \mathbb{F}_q have a group size of approximately q^g elements, they can be defined over the field $\mathbb{F}_{q'}$, where $q' \sim \sqrt[q]{q}$, to attain the same group size as an elliptic curve defined over \mathbb{F}_q . However, this also implies that the genus of a curve should be taken into account when assessing the security afforded by an embedding degree k . This is due to the MOV/FR attack, which uses the Tate pairing or the Weil pairing to transfer the DLP to the group $\mathbb{F}_{q^k}^*$. The so-called security parameter is then defined as the embedding degree divided by the genus of the curve

Galbraith [32] gives a bound $k(g)$ on the embedding degree of supersingular Abelian varieties of dimension g over \mathbb{F}_q . This bound depends solely on the genus, and not on the Abelian variety itself. For example, for supersingular Abelian varieties of dimension 2 the bound is $k(2) = 12$, and for supersingular Abelian varieties of dimension 3, the bound is $k(3) = 30$. As the embedding degree must be divided by the genus to give a more accurate estimation of the security, low genus supersingular hyperelliptic curves cannot give much more security from the MOV/FR attack than supersingular elliptic curves. However, Rubin and Silverberg [96] show that Galbraith's bounds are not achieved by simple Abelian varieties of dimension $g \geq 3$. An Abelian variety is simple if it does not decompose as a product of lower dimension Abelian varieties. As it is essential to work in large prime-order subgroups in cryptography, splitting Abelian varieties are not interesting. For the dimension 3 example, the actual bound that can be attained is $k(3) = 18$ (i.e. again $k/g = 6$).

Table 2.2 The maximum embedding degrees of supersingular hyperelliptic curves over \mathbb{F}_q [96]

genus (g)	1	2	3
q arbitrary (Galbraith's bounds)	6	12	30
q square	3	6	9
q nonsquare, $p = 2$	4	12	*
q nonsquare, $p = 3$	6	4	18
q nonsquare, $p > 3$	2	6	14

It is important to note that supersingular hyperelliptic curves of genus g may not be known with the maximum embedding degree as given by Rubin and Silverberg. For example, there is no known example of a supersingular hyperelliptic curve of genus 2 over \mathbb{F}_p with an embedding degree of $k = 6$, only an embedding degree of $k = 4$. The maximum embedding degrees for supersingular hyperelliptic curves of small genus are summarised in Table 2.2. Supersingular hyperelliptic curves of genus 3 with embedding degree 14 only exist in characteristic $p = 7$, not for large p , and so this case is not interesting for cryptography. As can be seen in Table 2.2, supersingular hyperelliptic curves only attain large embedding degrees in small characteristic. However, due to Coppersmith [21] a subexponential attack exists on the DLP in the finite field $\mathbb{F}_{2^{mk}}^*$ which is faster than in $\mathbb{F}_{p^k}^*$. This is an argument against using supersingular curves for pairing based cryptography, as opposed to ordinary curves over \mathbb{F}_p , as larger finite field parameters must be used.

2.4 Implementing Hyperelliptic Curve Cryptography

As explained in the previous chapter, only hyperelliptic curves of genus 1 (elliptic curves) and hyperelliptic curves of genus 2 are interesting for implementing cryptosystems that are based on the intractability of the DLP. In this section, more details are given on how these curves are used in cryptography.

2.4.1 Elliptic curves

A non-singular elliptic curve E defined over a field K is given by an equation of the form

$$E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

where $a_1, a_2, a_3, a_4, a_6 \in K$. As detailed in the previous section, the elements of the group $\text{Pic}_C^0(\mathbb{F}_q)$ of a hyperelliptic curve C of genus g can be represented by a semi-reduced divisor $D = \sum m_i(P_i) - (\sum m_i)(\infty)$, with the property that $\sum m_i \leq g$. For an elliptic curve, this means that D has only a single finite point $P_0 \in E(\mathbb{F}_q)$ in its support with $m_0 = 1$, such that D is of the form $D = (P_0) - (\infty)$. Therefore, there is a one-to-one correspondence between elements of $\text{Pic}_E^0(\mathbb{F}_q)$ and the points in $E(\mathbb{F}_q)$, where $[1, 0]$ is equivalent to the point at infinity ∞ . In other words, $E(\mathbb{F}_q)$ is isomorphic to $\text{Pic}_E^0(\mathbb{F}_q)$. Therefore, it is possible to work solely with $E(\mathbb{F}_q)$, the \mathbb{F}_q -rational points on the curve, rather than use Mumford representation as for elements of $\text{Pic}_C^0(\mathbb{F}_q)$. Note that for a hyperelliptic curve of genus $g > 1$, the points on the curve do not form a group.

The point at infinity ∞ can be thought of as a point on the y -axis, which lies so far away from the x -axis, that any vertical line (i.e. $x = c$, where c is a constant) passes through it. Cantor's algorithm for performing group arithmetic in $\text{Pic}_C^0(\mathbb{F}_q)$ corresponds exactly to the geometric chord and tangent operation on $E(\mathbb{F}_q)$, which is described as follows.

Definition 20 *Let $P, Q \in E(\mathbb{F}_q)$, l be the line connecting P and Q (or tangent line to E if $P = Q$) and R be the third point of intersection of l with E . Let v be the vertical line connecting R and ∞ . Then $P + Q$ is the point such that v intersects E at R, ∞ and $P + Q$.*

This definition easily yields explicit formulae for the group law that depend on the coordinates of the input points. These formulae are far simpler and faster to implement than the generic algorithm due to Cantor for arithmetic in $\text{Pic}_C^0(\mathbb{F}_q)$. Many different coordinate systems are available with which to perform group arithmetic. The advantage of using alternatives to the standard affine coordinates is to eliminate expensive field inversions, or to reduce the cost of doubling a point, which is the most prevalent operation in scalar

Table 2.3 Cost of group arithmetic for elliptic curves in \mathbb{F}_p

Coordinate System	Addition	Doubling
Affine Coordinates	$I, 2M, S$	$I, 2M, 2S$
Projective Coordinates	$12M, 2S$	$7M, 5S$
Jacobian Coordinates	$12M, 4S$	$4M, 6S$
Chudnovsky Jacobian Coordinates	$11M, 3S$	$5M, 6S$
Modified Jacobian Coordinates	$13M, 6S$	$4M, 4S$
Montgomery Scalar Multiplication	$9M, 2S$	$6M, 3S$

Table 2.4 Cost of group arithmetic for elliptic curves in \mathbb{F}_{2^m}

Coordinate System	Addition	Doubling
Affine Coordinates	$I, 2M, S$	$I, 2M, S$
Projective Coordinates	$16M, 2S$	$8M, 4S$
Jacobian Coordinates	$16M, 3S$	$5M, 5S$
Lopez-Dahab Coordinates	$13M, 4S$	$5M, 4S$
Montgomery Scalar Multiplication	$4M, 1S$	$2M, 3S$

multiplication. The cost of the group operation using different coordinates in \mathbb{F}_p is given in Table 2.3, and in \mathbb{F}_{2^m} in Table 2.4. Note that I, M, S denote a field inversion, multiplication and squaring, respectively. A clear conclusion to be drawn from Table 2.3 is that affine coordinates in \mathbb{F}_p should be avoided, as the ratio I/M is large. The situation is not so clear in \mathbb{F}_{2^m} as field inversion is not as expensive as in \mathbb{F}_p .

The set of points on an elliptic curve forms a finite Abelian group which meets all of the requirements for implementing a DLP-based cryptosystem. It has the added advantage of having an extremely simple representation and algorithm to compute the group law. The group $E(\mathbb{F}_q)$ has thus far resisted any successful attempt to apply the index-calculus. As a result, it is possible to use smaller field sizes for elliptic curve cryptography than for finite field cryptography. It remains to determine the cardinality of the group. The Hasse-Weil bound on the cardinality of $\text{Pic}_C^0(\mathbb{F}_q)$ for a general hyperelliptic curve C over \mathbb{F}_q is

$$(\sqrt{q} - 1)^{2g} \leq \#\text{Pic}_C^0(\mathbb{F}_q) \leq (\sqrt{q} + 1)^{2g}$$

In the elliptic curve case this simplifies to

$$(q - 2\sqrt{q} + 1) \leq \#E(\mathbb{F}_q) \leq (q + 2\sqrt{q} + 1),$$

from which the identity $\#E(\mathbb{F}_q) = q + 1 - t$ is obtained, where $|t| \leq 2\sqrt{q}$. The value t is called the trace of the Frobenius endomorphism. The characteristic polynomial of the Frobenius endomorphism for an elliptic curve E over \mathbb{F}_q is $\chi_E(T) = T^2 + a_1T + q$. The cardinality of $E(\mathbb{F}_q)$ can be evaluated with $\chi_C(1) = 1 + a_1 + q$, where $a_1 = -t$. Once $\#E(\mathbb{F}_q)$ is known, it is trivial to find $\#E(\mathbb{F}_{q^r})$ for some integer $r \geq 1$. Let $\chi_E(T) = (T - \alpha_0)(T - \alpha_1)$, then

$$\#E(\mathbb{F}_{q^r}) = q^r + 1 - \alpha_0^r - \alpha_1^r,$$

where α_0 and α_1 are conjugates. Several algorithms exist to compute the group order of an elliptic curve E defined over a large prime field \mathbb{F}_p in polynomial time, the first of which was given by Schoof [101].

In the previous section, an Abelian variety was defined as supersingular if it is isogenous to a product of supersingular elliptic curves (over $\overline{\mathbb{F}_q}$). Clearly, this definition is incomplete without defining supersingularity in the context of elliptic curves.

Definition 21 *An elliptic curve E over $\mathbb{F}_q = \mathbb{F}_{p^m}$ is supersingular if and only if $t \equiv 0 \pmod{p}$ where t is the trace of the Frobenius. Otherwise the curve is ordinary.*

In other words, for a curve to be supersingular the characteristic p must divide the trace of the Frobenius t . This can only happen if $\#E(\mathbb{F}_q) \equiv 1 \pmod{p}$. If $p \geq 5$, then E is supersingular over \mathbb{F}_p only if the trace of the Frobenius t equals zero, in which case $\#E(\mathbb{F}_p) = p + 1$. This yields an extremely fast method to evaluate a scalar multiple of a point $P = (x, y)$. As the trace is zero, the characteristic polynomial of the Frobenius is $\chi_E(\phi_p) = \phi_p^2 + p = 0$. Therefore $[p](x, y) = -\phi_p^2(x, y) = (x^{p^2}, -y^{p^2})$, which is extremely efficient to compute, as raising an element in \mathbb{F}_p to the power of p^2 is a linear operation.

This technique can be used in scalar multiplication by writing the scalar to the base p and using multi-exponentiation

The set of torsion points are the points whose order is finite, which is the case for all points on E over \mathbb{F}_q . The set of n -torsion points is defined as follows

$$E[n] = \{P \mid P \in E(\overline{\mathbb{F}_q}), [n]P = (\infty)\}$$

There are exactly n^2 n -torsion points. When p is the characteristic of \mathbb{F}_q , then a curve is ordinary if $E[p] \simeq \mathbb{Z}_p$, and supersingular if $E[p] \simeq 0$. In other words, supersingular curves have no finite points of order p with coordinates in $\overline{\mathbb{F}_q}$.

2.4.2 Genus 2 curves

A non-singular (imaginary quadratic) hyperelliptic curve C of genus 2 over a field K is given by the equation

$$C: y^2 + h(x)y = f(x),$$

where $h(x)$ is a polynomial in K of degree at most 2, and $f(x)$ is a monic polynomial in K of degree 5. All genus 2 curves are hyperelliptic, which is not the case for curves of genus $g > 2$. There is no isomorphism between $\text{Pic}_C^0(\mathbb{F}_q)$ and $C(\mathbb{F}_q)$, unlike in the elliptic case, and therefore $C(\mathbb{F}_q)$ does not form a group. The elements of $\text{Pic}_C^0(\mathbb{F}_q)$ are represented by a reduced divisor $D = \sum m_i(P_i) - (\sum m_i)(\infty)$, where $\sum m_i \leq 2$. It is possible to enumerate all the different types of reduced divisors that arise for genus 2 curves, $D = (0)$, $D = (P_0) - (\infty)$, $D = 2(P_0) - 2(\infty)$ and $D = (P_0) + (P_1) - 2(\infty)$. Divisors with a single finite point in the support are called degenerate divisors, and will prove to be extremely useful in pairing computation later in this thesis. Elements of $\text{Pic}_C^0(\mathbb{F}_q)$ for genus 2 curves are represented as $[x^2 + u_1x + u_0, v_1x + v_0]$, using the notation of Mumford.

It is relatively inefficient to use Cantor's algorithm to compute the group law for genus 2 curves, as it is a generic algorithm designed to cater for all possible curve equations and

Table 2.5 Cost of group arithmetic for genus 2 curves in \mathbb{F}_p ,

Coordinate System	Addition	Doubling
Affine Coordinates	$I, 22M, 3S$	$I, 22M, 5S$
Projective Coordinates	$47M, 4S$	$38M, 6S$
New Coordinates	$47M, 7S$	$34M, 7S$

genera. A large amount of work has been done on deriving explicit formulae from Cantor’s algorithm for genus 2 curves. It is only necessary to derive formulae for the most common case, which is when the divisors have maximal degree 2 and the finite points in the support all have different x -coordinates. Cantor’s algorithm can then be used to handle the other rarely-occurring cases. This work culminated in Lange’s formulae [70] for the affine case. Miyamoto et al. [85] introduced projective coordinates, which Lange [68] improved and extended to even characteristic. Lange [69] also introduced “new” coordinates as a generalisation of elliptic Jacobian, Chudnovsky Jacobian and modified Jacobian coordinates from elliptic curves to hyperelliptic curves of genus 2.

The cost of doubling and addition using the explicit formulae and various coordinate systems is given in Table 2.5 in \mathbb{F}_p and Table 2.6 in \mathbb{F}_{2^m} . Genus 2 curves over \mathbb{F}_{2^m} can be classified depending on the $h(x)$ polynomial. The cost in Table 2.6 assumes that $h(x) = h_2x^2 + h_1x + h_0$, where the leading coefficient $h_2 \neq 0$. Large speedups can be obtained when some of the coefficients of $h(x)$ are equal to 0. For example, curves of genus 2 where $h(x)$ is constant are supersingular, and extremely efficient explicit formulae can be derived for the group operation. Examining the tables, it is clear that inversion must be costly to justify using the alternative coordinate systems over affine coordinates. Using the ratios of I/M and S/M over \mathbb{F}_p and \mathbb{F}_{2^m} in Table 2.1, affine coordinates are more efficient for \mathbb{F}_{2^m} as well as in all but the most inefficient parameters for \mathbb{F}_p . Recently, Gaudry [38] and others have extended Montgomery scalar multiplication to the genus 2 case, which promises to be much more efficient than the coordinate systems given in Tables 2.5 and 2.6.

Therefore, genus 2 curves are a good alternative to using elliptic curves for DLP-based cryptography. Group elements can be represented in a reasonably compact manner as 4

Table 2.6 Cost of group arithmetic for genus 2 curves in \mathbb{F}_{2^m}

Coordinate System	Addition	Doubling
Affine Coordinates	$1, 22M, 3S$	$1, 20M, 6S$
Projective Coordinates	$49M, 4S$	$38M, 7S$
New Coordinates	$48M, 4S$	$39M, 6S$

elements of \mathbb{F}_q , and the explicit formulae for performing the group law are far more efficient than using Cantor's algorithm. As with elliptic curves, no algorithm exists to date for attacking the DLP in $\text{Pic}_C^0(\mathbb{F}_q)$ with complexity lower than the generic Pollard-rho algorithm. The Hasse-Weil bound on the order of $\text{Pic}_C^0(\mathbb{F}_q)$ for genus 2 curves implies that $\#\text{Pic}_C^0(\mathbb{F}_q) \approx q^2$. However, the size of the group for elliptic curves over \mathbb{F}_q is only around q . This means that a genus 2 curve can be defined over a finite field of \sqrt{q} elements to maintain the same security as an elliptic curve defined over a finite field of q elements. Smaller field sizes lead to more efficient implementations, particularly if a field element can fit inside a hardware register. However, this must be balanced against the more difficult arithmetic required in the genus 2 case.

As with elliptic curves, the Frobenius endomorphism can be used to speed up scalar multiplication for genus 2 Koblitz curves over \mathbb{F}_{2^m} , e.g. see Gunther et al. [44]. The characteristic polynomial of the Frobenius endomorphism for a genus 2 curve C over \mathbb{F}_q is given as

$$\chi_C(T) = T^4 + a_1T^3 + a_2T^2 + a_1qT + q^2 = \prod_{i=1}^4 (T - \alpha_i)$$

where $a_1, a_2 \in \mathbb{Z}$, and the α_i are complex numbers of absolute value \sqrt{q} . The group order over \mathbb{F}_{q^r} , for some integer $r \geq 1$, can be computed as

$$\#\text{Pic}_C^0(\mathbb{F}_{q^r}) = \prod_{i=1}^4 (1 - \alpha_i^r)$$

The group order over \mathbb{F}_q can be computed by determining the coefficients a_1 and a_2 of $\chi_C(T)$, and then evaluating $\chi_C(1)$. These coefficients are given as $a_1 = \#C(\mathbb{F}_q) - q - 1$

and $a_2 = (\#C(\mathbb{F}_{q^2}) - q^2 - 1 + a_1^2)/2$. Using this method to compute the group order over an extension of \mathbb{F}_q involves determining the α_i by factoring $\chi_C(T)$. However, this technique is only practical for curves defined over small fields. Numerous algorithms exist to compute $\#\text{Pic}_C^0(\mathbb{F}_q)$ for genus 2 curves in a more general way, see chapter 17 of Cohen et al [19] for more details.

The set of n -torsion divisors, $\text{Pic}_C^0[n]$, is given as

$$\text{Pic}_C^0[n] = \{D \mid D \in \text{Pic}_C^0(\overline{\mathbb{F}_q}), [n]D = (0)\}$$

$\text{Pic}_C^0[n]$ has group structure $(\mathbb{Z}/n\mathbb{Z})^4$ in the genus 2 case. As with elliptic curves, supersingular genus 2 curves have no divisor D of order p in $\overline{\mathbb{F}_q}$. See Galbraith [32] for more details on supersingular (genus 2) curves in cryptography.

2.5 The Tate Pairing

The Tate pairing (also called the Tate-Lichtenbaum pairing) was introduced to cryptography by Frey and Ruck [30], as an alternative bilinear pairing to the Weil pairing. In the context of pairing based cryptography, the Tate pairing is a pairing of Jacobian varieties defined over a finite field. Let C be a hyperelliptic curve defined over a finite field \mathbb{F}_q , and let n be a (large) prime such that $n \mid \#\text{Pic}_C^0(\mathbb{F}_q)$. n is also required to be co-prime to q to avoid the attack of Ruck [98]. Let k be the embedding degree as defined previously, and let $D_1 \in \text{Pic}_C^0(\mathbb{F}_{q^k})[n]$, such that $[n]D_1 = (f)$, for some function f . Let $D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})/n\text{Pic}_C^0(\mathbb{F}_{q^k})$. To ensure a non-trivial pairing value, D_1 and D_2 must have disjoint support.

Definition 22 *The Tate pairing of level n is a map*

$$\langle \cdot, \cdot \rangle_n : \text{Pic}_C^0(\mathbb{F}_{q^k})[n] \times \text{Pic}_C^0(\mathbb{F}_{q^k})/n\text{Pic}_C^0(\mathbb{F}_{q^k}) \rightarrow \mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^n$$

and is defined as

$$\langle D_1, D_2 \rangle_n = f(D_2)$$

The Tate pairing satisfies the following properties

- 1 (Bilinearity) $\langle [a]D_1, [b]D_2 \rangle_n = \langle D_1, D_2 \rangle_n^{ab}$ for all $a, b \in \mathbb{Z}$ (modulo n th powers)
- 2 (Non-degeneracy) For each divisor $D_1 \in \text{Pic}_C^0(\mathbb{F}_{q^k})[n]$ $D_1 \neq (0)$ there is some divisor $D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})/n\text{Pic}_C^0(\mathbb{F}_{q^k})$ such that $\langle D_1, D_2 \rangle \notin (\mathbb{F}_{q^k}^*)^n$
- 3 (Computability) $\langle D_1, D_2 \rangle_n$ can be efficiently computed

The subscript n in $\langle \cdot, \cdot \rangle_n$ can be dropped if it is obvious from the context. The first input to the Tate pairing is an element of $\text{Pic}_C^0(\mathbb{F}_{q^k})[n]$, the group of n -torsion divisors in \mathbb{F}_{q^k} . However, the second input is an element of $\text{Pic}_C^0(\mathbb{F}_{q^k})/n\text{Pic}_C^0(\mathbb{F}_{q^k})$, where $n\text{Pic}_C^0(\mathbb{F}_{q^k}) = \{[n]D \mid D \in \text{Pic}_C^0(\mathbb{F}_{q^k})\}$ is a subgroup of $\text{Pic}_C^0(\mathbb{F}_{q^k})$. The quotient group $\text{Pic}_C^0(\mathbb{F}_{q^k})/n\text{Pic}_C^0(\mathbb{F}_{q^k})$ is the set of equivalence classes of elements in $\text{Pic}_C^0(\mathbb{F}_{q^k})$ under the equivalence relation $D_1 \sim D_2$ such that $(D_1 - D_2) \in n\text{Pic}_C^0(\mathbb{F}_{q^k})$.

Let n be a prime as defined, and let $\text{Pic}_C^0(\mathbb{F}_q)$ have no elements of order n^2 . In other words, n^2 should not divide $\#\text{Pic}_C^0(\mathbb{F}_q)$. Then the group $\text{Pic}_C^0(\mathbb{F}_{q^k})/n\text{Pic}_C^0(\mathbb{F}_{q^k})$ is isomorphic to $\text{Pic}_C^0(\mathbb{F}_{q^k})[n]$. Therefore, the second input to the Tate pairing can be taken to be an element of the n -torsion group. However, it is unnecessary to restrict D_2 to this group, as the output of the Tate pairing is not affected by the choice of $D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})$ as representative of the class. Therefore, the Tate pairing is defined as the simplified map

$$\text{Pic}_C^0(\mathbb{F}_{q^k})[n] \times \text{Pic}_C^0(\mathbb{F}_{q^k}) \rightarrow \mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^n$$

The output of the Tate pairing is an element of the quotient group $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^n$. Let $\mu_n = \{u \in \mathbb{F}_{q^k}^* \mid u^n = 1\}$ be the group of the n th roots of unity, and define $(\mathbb{F}_{q^k}^*)^n = \{u^n \mid u \in \mathbb{F}_{q^k}^*\}$, which is a subgroup of $\mathbb{F}_{q^k}^*$. The quotient group $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^n$ is isomorphic to μ_n . Two elements $a, b \in \mathbb{F}_{q^k}^*$ are equivalent (when considered as elements of $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^n$)

if $a/b \in (\mathbb{F}_{q^k}^*)^n$. In other words, $a = bc^n$ for some $c \in \mathbb{F}_{q^k}^*$. As the output of the Tate pairing is defined up to a multiple by the n th power of $c \in \mathbb{F}_{q^k}^*$, it is necessary to modify the output to obtain a unique value suitable for cryptography. An obvious way to remove the n th powers is to exponentiate the pairing value to the power of $(q^k - 1)/n$, as $(c^n)^{(q^k-1)/n} = c^{(q^k-1)} = 1$. This exponentiation is known as the “final exponentiation”, and the resulting pairing is known as the reduced Tate pairing.

Definition 23 *The reduced Tate pairing is defined as*

$$\langle D_1, D_2 \rangle_n^{(q^k-1)/n} = f(D_2)^{(q^k-1)/n} \in \mu_n$$

The non-degeneracy property of the (reduced) Tate pairing states that for a given divisor $D_1 \in \text{Pic}_C^0(\mathbb{F}_{q^k})[n]$, apart from $D_1 = (0)$, there is a divisor $D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})$ such that $\langle D_1, D_2 \rangle \neq 1$. However, a method is needed to construct D_2 such that the pairing is non-degenerate. As will be detailed in the following chapter, instead of defining $D_1 \in \text{Pic}_C^0(\mathbb{F}_{q^k})[n]$, it is common in practice to define D_1 over \mathbb{F}_q , i.e. $D_1 \in \text{Pic}_C^0(\mathbb{F}_q)[n]$. If D_2 is also defined over \mathbb{F}_q and $k > 1$ then the pairing value is degenerate, as it will be eliminated by the final exponentiation. Two techniques are known to construct D_2 to guarantee non-degeneracy in the case that $D_1 \in \text{Pic}_C^0(\mathbb{F}_q)$. The first technique uses distortion maps on supersingular curves, and the second uses trace maps on ordinary curves.

Verheul [113] introduces distortion maps and shows how to apply them to pairing computation. Let $D \in \text{Pic}_C^0(\mathbb{F}_q)$ be a non-trivial divisor with prime order n , such that n^2 does not divide $\#\text{Pic}_C^0(\mathbb{F}_q)$, and let the embedding degree $k > 1$. A distortion map ψ is then a non-rational endomorphism which maps $D \in \text{Pic}_C^0(\mathbb{F}_q)$ to $D' \in \text{Pic}_C^0(\mathbb{F}_{q^k}) \setminus \text{Pic}_C^0(\mathbb{F}_q)$. Distortion maps are used to guarantee non-degeneracy, as the so-called modified pairing of random elements D' in a specific subgroup of order n in $\text{Pic}_C^0(\mathbb{F}_{q^k})$.

Definition 24 *Let $D_1, D_2 \in \text{Pic}_C^0(\mathbb{F}_q)[n]$. Then the modified pairing is defined as*

$$\langle D_1, \psi(D_2) \rangle_n^{(q^k-1)/n} \neq 1$$

The modified pairing is guaranteed to be non-degenerate as the distorted divisor $D'_2 = \psi(D_2)$ is linearly independent from D_1 . Another advantage to using distortion maps is that they facilitate the generation of random elements $D' \in \text{Pic}_C^0(\mathbb{F}_{q^k})$, which can be done by simply generating a random $D \in \text{Pic}_C^0(\mathbb{F}_q)$ and then using ψ to map D into the larger field. As distortion maps always exist for supersingular curves, the modified Tate pairing is commonly used in practice.

Distortion maps do not exist on ordinary curves, and hence a different technique must be used in this case to guarantee non-degeneracy. Let $D_1 \in \text{Pic}_C^0(\mathbb{F}_{q^k})$. The trace map on D_1 is defined as

$$\text{Tr}(D_1) = \sum_{i=1}^k \phi_{q^i}(D_1) \in \text{Pic}_C^0(\mathbb{F}_q),$$

where ϕ_{q^i} is the q^i -th power Frobenius endomorphism. The trace map can be used to guarantee a non-degenerate pairing in the following way. Let $D_1, D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})[n]$, $D_1, D_2 \notin \text{Pic}_C^0(\mathbb{F}_q)$ and $\text{Tr}(D_1) \neq (0)$. Then

$$\langle \text{Tr}(D_1), D_2 \rangle_n^{(q^k-1)/n} \neq 1$$

Another bilinear pairing is the Weil pairing. An additional constraint on the prime subgroup order $n \nmid \#\text{Pic}_C^0(\mathbb{F}_q)$, n co-prime to q , is that n must also be co-prime to $q - 1$. This condition ensures that the Weil pairing is efficiently computable. Let $D_1, D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})[n]$, where D_1 and D_2 are in distinct cyclic subgroups of order n . Let f be a function such that $(f) = [n]D_1$, and g be a function such that $(g) = [n]D_2$. Again, to ensure a non-trivial pairing value, D_1 and D_2 must have disjoint support.

Definition 25 *The Weil pairing is a map*

$$e_n(\cdot, \cdot) : \text{Pic}_C^0(\mathbb{F}_{q^k})[n] \times \text{Pic}_C^0(\mathbb{F}_{q^k})[n] \rightarrow \mu_n,$$

and is defined as

$$e_n(D_1, D_2) = \frac{f(D_2)}{g(D_1)}$$

The Weil pairing has the following properties

- 1 (Bilinearity) $e_n([a]D_1, [b]D_2) = e_n(D_1, D_2)^{ab}$ for all $a, b \in \mathbb{Z}$
- 2 (Non-degeneracy) For each divisor $D_1 \in \text{Pic}_C^0(\mathbb{F}_{q^k})[n]$, $D_1 \neq (0)$ there is some divisor $D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})[n]$ such that $e_n(D_1, D_2) \neq 1$
- 3 (Computability) $e_n(D_1, D_2)$ can be efficiently computed
- 4 (Alternating) For all $D_1, D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})[n]$ then $e_n(D_1, D_2) = e_n(D_2, D_1)^{-1}$
- 5 (Compatibility) If $D_1 \in \text{Pic}_C^0(\mathbb{F}_{q^k})[nm]$ and $D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})[n]$ then $e_{nm}(D_1, D_2) = e_n([m]D_1, D_2)$
- 6 If $\phi: E \rightarrow E'$ is an isogeny with dual $\hat{\phi}$ then $e_n(\phi(P), Q) = e_n(P, \hat{\phi}(Q))$

The Weil pairing can be computed using two applications of the Tate pairing such that

$$e_n(D_1, D_2) = \frac{\langle D_1, D_2 \rangle_n}{\langle D_2, D_1 \rangle_n},$$

where the equivalence is up to n th powers. Note that this implies that $e_n(D_1, D_1) = 1$, which is not necessarily the case for the Tate pairing. When assessing which is the more efficient pairing, the comparison is between the final exponentiation to $(q^k - 1)/n$ required for the (reduced) Tate pairing, and the computation of $\langle D_2, D_1 \rangle_n$ required for the Weil pairing. This question will be explored further in the following chapter, however it suffices to say for the moment that the Tate pairing can be computed more efficiently. Therefore, the Tate pairing is the most interesting pairing for cryptographic applications.

A bilinear pairing is said to be symmetric if swapping the arguments yields the same pairing output. As the Weil pairing can be calculated with two applications of the Tate

pairing, it follows that the Tate pairing is not symmetric, as this would violate the non-degeneracy property of the Weil pairing. However, both the reduced Tate pairing and the Weil pairing are symmetric when a distortion map is applied to the second argument, or the trace map is applied to the first argument, as both are then restricted to a cyclic subgroup. For example, let $D_2 = [m]D_1$ and let ψ be a distortion map. Then

$$e_n(D_1, \psi(D_2)) = e_n(D_1, [m]\psi(D_1)) = e_n([m]D_1, \psi(D_1)) = e_n(D_2, \psi(D_1))$$

Therefore, as supersingular hyperelliptic curves of genus 2 are considered in this thesis, the property of symmetry, as required in the definition of a bilinear pairing in Chapter 1, is provided by restricting both input elements to the pairing to the same group $\text{Pic}_C^0(\mathbb{F}_q)[m]$ and using a distortion map.

The computability property of the Tate pairing simply states that the Tate pairing can be computed efficiently. The main task involved in computing the Tate pairing $\langle D_1, D_2 \rangle_n$, is to construct the function f such that $(f) = [n]D_1$. In an unpublished manuscript in 1986 by Miller [82] (and later published in 2004 as Miller [84]), it was shown how to efficiently construct this function in stages by using a double-and-add algorithm. This algorithm, known universally as Miller's algorithm, was originally used to compute the Weil pairing in polynomial time. However, it can be easily adapted to compute the Tate pairing.

Let $D_1, D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})$, and let $D_3 \in \text{Pic}_C^0(\mathbb{F}_{q^k})$ be the divisor that is formed from the Cantor composition and reduction of D_1 and D_2 . Then

$$D_1 + D_2 - D_3 = (f) = (c/d),$$

where $f = c/d$ is a function on C which is independent of the choice of the representatives for D_1, D_2 and D_3 . The goal is to construct a function f_n , such that $(f_n) = [n]D_1$. Let $f_1 = 1$, and let f_i be the function appearing in

$$(f_i) = (i)D - [i]D,$$

where $(\iota)D$ stands for the (symbolic) addition of the divisor ι times to itself, while $[\iota]D$ stands for the reduced result. Then $f_{\iota+j}$ is defined as

$$\begin{aligned} (f_{\iota+j}) &= (\iota+j)D - [\iota+j]D \\ &= (\iota)D - [\iota]D + (j)D - [j]D - [\iota+j]D + [\iota]D + [j]D \\ &= (f_{\iota}f_j \frac{c}{d}) \end{aligned}$$

If $\iota = j$, then the addition is replaced by a doubling, and f_{ι}^2 appears in the function. Therefore, the function f_n can be constructed in stages by using a double-and-add algorithm. The actual value for the function f_n is not of interest, rather it is the value of f_n at a divisor D that is required. Each iteration of the algorithm, the rational functions c and d are evaluated at the image divisor $D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})$. This is possible due to the fact that only multiplication, and not addition, is required to compute $f_{\iota+j}$. This result is then multiplied with an accumulating variable $f \in \mathbb{F}_{q^k}$, which must also be squared each time the iterating divisor is doubled. The resulting algorithm is known as Miller's algorithm, and is given in Algorithm 3.

If any of the intermediate functions in Miller's algorithm has a zero at the evaluating divisor D_2 , the algorithm will fail. There are a number of techniques to ensure that this does not happen. One strategy is to evaluate at the divisor $D'_2 = (D_2 + D') - (D')$, rather than evaluate at D_2 , where D' is a randomly chosen element of $\text{Pic}_C^0(\mathbb{F}_{q^k})$. This technique involves evaluating the rational functions c and d at the divisors $(D_2 + D')$ and D' at each iteration, and multiplying the the accumulating function by the result. However, this is quite inefficient and techniques will be described in the following chapter to avoid this penalty.

It remains to examine how to derive the necessary intermediate functions c and d for the two cases of interest for cryptography, namely elliptic and genus 2 curves. The intermediate functions are calculated implicitly as part of the composition and reduction process on the iterating divisor. In the case of elliptic curves, the functions are simply the straight lines used in the addition process. So when adding two points $P, Q \in E(\mathbb{F}_{q^k})$, the function c is

Algorithm 3 Miller's algorithm to compute the Tate pairing

INPUT $D_1 \in \text{Pic}_C^0(\mathbb{F}_{q^k})[n], D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})$
OUTPUT $\langle D_1, D_2 \rangle_n$

- 1 $f \leftarrow 1$
- 2 $T \leftarrow D$
- 3 $D'_2 \leftarrow (D_2 + D') - (D')$
- 4 **for** $i \leftarrow \lfloor \log_2(n) \rfloor - 1$ **downto** 0 **do**
- 5 \triangleright Compute T', c, d such that $T' = (2)T - (c/d)$
- 6 $f \leftarrow f^2 \frac{c(D'_2)}{d(D'_2)}$
- 7 $T \leftarrow T'$
- 8 **if** $n_i = 1$ **then**
- 9 \triangleright Compute T', c, d such that $T' = T + D - (c/d)$
- 10 $f \leftarrow f \frac{c(D'_2)}{d(D'_2)}$
- 11 $T \leftarrow T'$
- 12 **end if**
- 13 **end for**
- 14 **Return** f

the straight line through P and Q , and the function d is the vertical line through the point $P + Q$. In this thesis, we follow the convention of calling the function c the “line function”, and d the “vertical line function”. However, it must be emphasised that this is not strictly accurate for higher genus curves.

For genus 2 curves it is not quite so straightforward. Let $D_1, D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})$ be the two divisors that are being added, given in Mumford representation as $D_1 = [u_1, v_1]$ and $D_2 = [u_2, v_2]$. In the composition stage of Cantor's algorithm, the polynomial $\delta = \gcd(u_1 - u_2, v_1 + v_2 + h)$ is computed. Now let $D_3 = [u_3, v_3]$ be the output of the Cantor composition algorithm on D_1 and D_2 , and let $D'_3 = [u'_3, v'_3]$ be the reduced divisor equivalent to D_3 . If the divisor D_3 is already reduced following the composition stage, then the function $f(x, y) = c(x, y)/d(x, y) = \delta(x)$. If this is not the case, then the function $f(x, y) = c(x, y)/d(x, y) = \delta(x)(y - v_3(x))/u'_3(x)$. In the overwhelmingly common case $\delta = 1$ and thus $c(x, y) = y - v_3(x)$ and $d(x, y) = u'_3(x)$.

As Miller's algorithm has $\lfloor \log_2(n) \rfloor$ iterations, there will be $\lfloor \log_2(n) \rfloor$ doublings. Also if n has a random Hamming weight, there will be $\lfloor \log_2(n)/2 \rfloor$ additions to be performed in the loop. In Algorithm 3 the iterating divisor T is an \mathbb{F}_{q^k} -rational divisor. Therefore, per-

forming doubling or addition on this divisor is computationally expensive, even assuming the use of explicit formulae. The rational functions that are extracted from the addition process for both doubling and addition are then evaluated at the image divisor $D_2 \in \text{Pic}_C^0(\mathbb{F}_{q^k})$. After this, the accumulating variable $f \in \mathbb{F}_{q^k}$ (also known as the Miller variable) must be updated by the result of evaluating c and d at D_2' . However, this operation is extremely expensive, as with a naive implementation an inversion is required over \mathbb{F}_{q^k} . Finally, the accumulating variable is squared whenever a doubling takes place, which again is not a cheap operation.

2.6 Conclusion

In this chapter, it has been described how finite field arithmetic can be implemented in an efficient manner using a polynomial basis. Hyperelliptic curves have been introduced, and their application to cryptography has been detailed. In particular, hyperelliptic curves of genus 1 and 2 have been shown to be suitable for implementation due to their compact representation, efficient explicit formulae for the group law, and lack of effective index calculus attacks.

The Tate pairing and Weil pairing have been defined in the context of hyperelliptic curves. It has been shown how to construct pairings to avoid trivial values, and an efficient algorithm due to Miller to compute pairings has been described. Both the Weil pairing and the Tate pairing are suited to implementing cryptographic protocols that are based on the intractability of the BDHP in $\text{Pic}_C^0(\mathbb{F}_{q^k})$.

Chapter 3

Optimisations to Miller's Algorithm

3.1 Introduction

In 1993, Menezes [78] reported an implementation of the Weil pairing that ran in “just a few minutes” on a SUN-2 SPARC-station using an elliptic curve over \mathbb{F}_{2^m} , where $m \approx 200$. This was when pairings were used to reduce the HCDLP on supersingular curves to the DLP in $\mathbb{F}_{q^k}^*$, and hence the speed of pairing computation was not particularly important. However, with the emergence of cryptographic protocols that are based on computing either the Weil or Tate pairings, it quickly became obvious that it was of paramount importance to improve the relatively slow computational speed of Miller's algorithm as originally described.

In this chapter, an overview of various methods to improve the performance of Miller's algorithm is provided. To the best of our knowledge, this is the first comprehensive review on this matter in the literature. The emphasis is mainly on elliptic curves, as nearly all of the improvements were derived in this context. However, a substantial number of the techniques can be generalised to the hyperelliptic case. Firstly, the early optimisations to Miller's algorithm are examined. These include defining the iterating point over a subfield, as well as modifying the image point in such a way that the calculation of vertical line functions can be omitted from the algorithm entirely.

Secondly, the concept of squared pairings is examined. Initially it appeared that these

pairings could be computed more efficiently than plain pairings. However, it was later shown that plain pairings always yield a more efficient implementation. Thirdly, the generalisation of pairing calculation to hyperelliptic curves of genus greater than one is detailed. Certain curves are shown to be extremely suited to pairing implementation, as the Tate pairing can be computed with a shortened loop size and a trivial final exponentiation. These curves also support a simple choice of function in the loop which eliminates the need for explicitly computing multiples of the iterating point.

Fourthly, the concept of pairing compression is detailed. Pairing compression involves modifying the output of Miller's algorithm to take up less bandwidth. Both the trace-based approach as well as the alternative method of using algebraic tori are examined. It is then shown how to compute the Weil pairing efficiently, using many of the optimisations defined earlier in the chapter. A metric for implementing finite field arithmetic in an efficient manner is also examined. This metric can be used to analyse the cost of pairing computation in a theoretical manner. Finally, a small number of recent optimisations are examined and the chapter is concluded.

3.2 Early Optimisations

Here the Tate pairing is considered in the elliptic case E , rather than in the more general hyperelliptic setting. Let $D_1 = (P) - (\infty) \in \text{Pic}_E^0(\mathbb{F}_{q^k})[n]$, such that $[n]D_1 = n(P) - n(\infty) = (f)$, and let $D_2 = (Q) - (\infty) \in \text{Pic}_E^0(\mathbb{F}_{q^k})$. Rather than compute the Tate pairing using D_1 and D_2 , it is more convenient to exploit the isomorphism between $\text{Pic}_E^0(\mathbb{F}_{q^k})$ and $E(\mathbb{F}_{q^k})$, and compute the pairing on P and Q . To ensure that none of the intermediate functions in Miller's algorithm have a zero at Q , the second argument to the Tate pairing is defined as $Q' = (Q + R) - (R)$, where $R \in E(\mathbb{F}_{q^k})$ is a random point on the curve such that $R \notin \{\infty, -P\}$. Therefore, the Tate pairing is computed as

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

Galbraith et al [31] introduce three improvements to Miller's algorithm for computing the Tate pairing. The main observation is that the first input point P in $\langle P, Q' \rangle_n$ should be defined over \mathbb{F}_q rather than over \mathbb{F}_{q^k} . This observation was previously made by Boneh and Franklin [11] in the case of the Weil pairing. Galbraith et al suggest representing the field \mathbb{F}_{q^k} as a degree k extension of the base field $\mathbb{F}_q = \mathbb{F}_{p^m}$, rather than working with extensions of \mathbb{F}_p of degree m , as suggested by Boneh et al [12]. However, arithmetic in \mathbb{F}_{q^k} is still far more expensive than in \mathbb{F}_q . If P is defined over \mathbb{F}_q , then the coefficients of the straight lines used in the addition process will also be defined over \mathbb{F}_q . This leaves only the evaluation of the line functions, and the subsequent multiplication by the accumulating Miller variable (which is also squared), to be performed in \mathbb{F}_{q^k} . This idea substantially reduces the computational cost of Miller's algorithm. Galbraith et al also show that the random point R in Miller's algorithm can be defined over \mathbb{F}_q instead of \mathbb{F}_{q^k} .

The second observation of Galbraith et al relates to removing inversions from the algorithm. The line functions must be divided by the vertical line functions at each iteration of the loop. However, inversion in the field \mathbb{F}_{q^k} is extremely expensive. The idea of Galbraith et al is to use two accumulating variables in Miller's algorithm, and to perform a single inversion after the loop. One variable keeps track of the numerator, or line functions, and the other keeps track of the denominator, or vertical line functions. Each variable must be squared whenever a point doubling is performed. This technique effectively trades an inversion for a squaring at each iteration of Miller's algorithm, which results in a dramatic improvement, as squaring is a relatively cheap operation in \mathbb{F}_{q^k} compared to inversion.

The third observation is that windowing methods can be used to compute Miller's algorithm. This observation was also made by Boneh et al [12]. Windowing methods are routinely used for scalar multiplication on elliptic curves. Given a point $P \in E(\mathbb{F}_q)$, the basic idea is to precompute the values $[z]P$ for all values z in a window of size 3 or 4 bits. Windowing methods reduce the number of additions that must be performed, but do not affect the number of doubling operations. However, in chapter 9 of Blake et al [9], Galbraith shows that using windowing methods to compute the Tate pairing is not efficient, as

there is an increase in the number of field multiplications in the addition stage of Miller's algorithm

Galbraith et al also discuss how to select the order n used to compute the Tate pairing. Let $H(n)$ be the Hamming weight of n . Then the number of additions to be performed in Miller's algorithm corresponds to $H(n) - 1$. Galbraith et al discuss how n can be chosen so that it has as low a Hamming weight as possible. It is also shown how the actual group order can be used, rather than a subgroup order. This is useful for certain supersingular curves in small characteristic, which have a group order of low Hamming weight, but do not have a large prime factor of low Hamming weight. In this case, the Tate pairing can be computed using the group order N , and the final exponentiation becomes $(q^k - 1)/N$, which also has a low Hamming weight. Note that while the result is still a unique n th root of unity, it may not be the same value as when the Tate pairing is computed with respect to the large prime n .

Some supersingular curves in low characteristic p have extremely efficient formulae to compute $[p^i]P$, where $i > 0$ is an integer and P is a point on the curve. Galbraith et al show how this property can be exploited in pairing computation. An example is given of two supersingular curves in characteristic 3 with an embedding degree of $k = 6$. These curves have a formula for computing $[3]P$ which does not involve inversion. As this formula is very efficient, it is natural to consider using it to compute the Tate pairing for these curves. This can be done by using a ternary basis in Miller's algorithm, instead of a binary basis as is standard. The accumulating variable must then be cubed each time a point tripling is performed, which can be computed efficiently in characteristic 3.

Barreto et al [5] also present an improved variant of Miller's algorithm to compute the Tate pairing. The most important contribution of this paper is a deterministic variant of Miller's algorithm to compute the Tate pairing, which is far less expensive to compute than the conventional algorithm. This algorithm depends on the first point P in $\langle P, Q' \rangle_n$ being defined over \mathbb{F}_q rather than over the larger field \mathbb{F}_{q^k} . As discussed, Galbraith et al [31] independently discovered the benefits of defining P over \mathbb{F}_q . However, Barreto et

Table 3 1 Some supersingular elliptic curves with low k

Curve equation	Finite field	Curve order	k
$E_{1,d} \quad y^2 = x^3 + x - dx + d, d \in \{0, 1\}$	$\mathbb{F}_p, p > 3$	$p + 1$	2
$E_{2,d} \quad y^2 + y = x^3 + x + d, d \in \{0, 1\}$	\mathbb{F}_{2^m}	$2^m \pm 2^{(m+1)/2} + 1$	4
$E_{3,d} \quad y^2 = x^3 - x + d, d \in \{-1, 1\}$	\mathbb{F}_{3^m}	$3^m \pm 3^{(m+1)/2} + 1$	6

al achieve further computational savings by introducing two new optimisations Barreto et al first of all show that $(q - 1)$ is a factor of $(q^k - 1)/n$ for the curves in Table 3 1, all of which have an even embedding degree k If the random point R is also defined over \mathbb{F}_q rather than \mathbb{F}_{q^k} , then by Fermat's little theorem $f(R)^{q-1} = 1$, and hence $f(R)^{(q^k-1)/n} = 1$ Therefore, the evaluation at R can be omitted altogether from Miller's algorithm, resulting in a deterministic algorithm that is computed as $\langle P, Q \rangle_n^{(q^k-1)/n} = f(Q)^{(q^k-1)/n}$

Barreto et al 's second contribution is the idea of 'denominator elimination' Recall that two functions are extracted from the addition process in Miller's algorithm In the elliptic case, the line function corresponds to the tangent at the iterating point (if doubling), or the line between two points (if adding) The vertical line function consists of the equation of the vertical line through the resulting point The line function must be divided by the vertical line function each time a doubling or addition takes place Galbraith et al [31] avoid this by using two accumulating variables and performing a single inversion after the loop However, Barreto et al show how to completely avoid computing the vertical line functions, which improves the speed of Galbraith et al 's algorithm by nearly 50%

As detailed in Chapter 2, the modified Tate pairing is typically used when implementing pairings using supersingular curves In the elliptic case, this involves generating a point $Q \in E(\mathbb{F}_q)$ and using a distortion map to obtain a point $\psi(Q) \in E(\mathbb{F}_{q^k})$, which can then be used as the image point in Miller's algorithm Table 3 2 gives a suitable distortion map for some of the curves defined previously in Table 3 1 Barreto et al show that the vertical line functions can be discarded when computing the modified Tate pairing using any of these distortion maps The key reason for this is that the distortion maps given in Table 3 2 map the x -coordinate of the point Q to a subfield of \mathbb{F}_{q^k} , whereas the y -coordinate

Table 3.2 Distortion maps for (most of) the curves given in Table 3.1

Curve	Finite field	Distortion map	Conditions
$E_{1,0}$	\mathbb{F}_p	$\psi_1(x, y) = (-x, iy)$	$p \equiv 3 \pmod{4}$, $i \in \mathbb{F}_{p^2}$, $i^2 = -1$
$E_{2,0}, E_{2,1}$	\mathbb{F}_{2^m}	$\psi_2(x, y) = (x + s^2, y + sx + t)$	$s, t \in \mathbb{F}_{2^{4m}}$, $s^4 + s = 0$, $t^2 + t + s^6 + s^2 = 0$
$E_{3,-1}, E_{3,1}$	\mathbb{F}_{3^m}	$\psi_3(x, y) = (-x + r_d, iy)$	$r_d \in \mathbb{F}_{3^{3m}}, i \in \mathbb{F}_{3^{2m}}$, $r_d^3 - r_d - d = 0$, $i^2 = -1$

is mapped to the full field \mathbb{F}_{q^k}

The vertical line functions are defined by an equation $x - x_{[i]P} = 0$, where $x_{[i]P}$ is the x -coordinate of some multiple i of P , and x is a variable which will be later evaluated at $x_{\psi(Q)}$ (the value of which remains constant throughout the algorithm). As P is defined over the base field \mathbb{F}_q , the x -coordinate of $x_{[i]P}$, a multiple of P , will also be defined over \mathbb{F}_q . In addition, the distortion map ψ leaves the x -coordinate of $\psi(Q)$ defined over a subfield of \mathbb{F}_{q^k} . Therefore, the vertical line functions in Miller's algorithm will not be defined over \mathbb{F}_{q^k} , but over some subfield. It can be shown that when i divides the embedding degree k , then $(q^i - 1)$ divides $(q^k - 1)$. This exponentiation eliminates all terms defined over \mathbb{F}_{q^i} , and thus there is no need to include the vertical line functions in Miller's algorithm.

The denominator elimination technique of Barreto et al. is approximately 50% faster than the algorithm of Galbraith et al. This is because the denominator elimination technique removes the need for the second variable, and hence saves a squaring and a multiplication in \mathbb{F}_{q^k} each time a doubling is performed in the loop. Barreto et al. also examine the use of prime order subgroups of low Hamming weight. They propose using a Solinas [109] prime as the subgroup order, which is a prime number of the form $p = 2^\alpha \pm 2^\beta \pm 1$. As a result, only two additions must be performed in Miller's algorithm, which can then be unrolled to remove conditional logic.

Barreto et al. give a technique to speed up the final exponentiation required for the

Tate pairing One way to evaluate this function is to precompute $(q^k - 1)/n$ and to use a square-and-multiply algorithm with windowing techniques to compute the exponentiation. However, Barreto et al. observe that for an even embedding degree k , the factor $(q^{k/2} - 1)$ can be extracted from the final exponentiation. This exponentiation can be evaluated with a single field inversion and multiplication in \mathbb{F}_{q^k} , as raising to the power of $q^{k/2}$ is trivially computed as a conjugation with respect to $\mathbb{F}_{q^{k/2}}$. The remaining exponentiation to $(q^{k/2} + 1)/n$ can sometimes be factored further, but remains an expensive operation to compute.

Izu and Takagi [51] investigate the computation of the Tate pairing using elliptic curves over large prime fields. Their main contribution is to evaluate the use of alternative coordinate systems in Miller's algorithm. As with scalar multiplication, the best coordinate system to use depends on the finite field. They also show how to optimise the generation of the coefficients of the line functions when random points are included in the algorithm. However, this optimisation is not useful in practice due to the deterministic algorithm to compute the Tate pairing given by Barreto et al. [5]. Izu and Takagi also investigate using an explicit formula to compute $[2^i]P$, instead of using the double-and-add approach.

Chatterjee et al. [15] also examine the implementation of the Tate pairing using elliptic curves over large prime fields. The main contribution of this paper is a method to encapsulate the computation of the line function with the doubling process on the iterating point P . Jacobian coordinates are used to represent P , and it is shown how some of the operations in the encapsulated method do not need to be calculated, as they are eliminated by the final exponentiation. This technique is also shown to apply in the addition stage of Miller's algorithm when mixed addition is used. Chatterjee et al. examine the memory requirements of Miller's algorithm, as well as showing how certain operations can be parallelised for implementation in hardware. It is also shown how to exploit the NAF to compute the Tate pairing.

The denominator elimination technique was previously defined for certain supersingular curves with distortion maps of a special form. Barreto et al. [6, 7] generalise this technique to ordinary elliptic curves, by removing the need for a distortion map. It is shown that when

k is even and $d|k$, then $q^d - 1$ is a factor of $q^k - 1$. This implies that the accumulating function can be multiplied by any nonzero element $r \in \mathbb{F}_{q^d}$ without changing the pairing output. Therefore, the denominator elimination technique can be generalised for the case that the x -coordinate of the image point Q is defined over some subfield \mathbb{F}_{q^d} of \mathbb{F}_{q^k} , where $d|k$. Note that the y -coordinate of Q must be defined over \mathbb{F}_{q^k} or else the entire pairing value will be defined over \mathbb{F}_{q^d} and will be eliminated by the final exponentiation.

Barreto et al. then give two techniques to show how denominator elimination can work in the absence of supersingular curves and suitable distortion maps. Let $d = k/2$ (and hence the embedding degree k is even). The first technique uses twists. Let $E: y^2 = x^3 + ax + b$ be an elliptic curve over the finite field \mathbb{F}_q , of characteristic $p > 3$. Then the quadratic twist of E over \mathbb{F}_{q^d} is $E': y^2 = x^3 + v^2ax + v^3b$, for some quadratic non-residue $v \in \mathbb{F}_{q^d}$. Let v be a quadratic residue in \mathbb{F}_{q^k} , then the map $\psi(x, y) \mapsto (v^{-1}x, (v\sqrt{v})^{-1}y)$ is an isomorphism that maps the group of points of $E'(\mathbb{F}_{q^d})$ to a subgroup of $E(\mathbb{F}_{q^k})$.

Now let Q' be a point on the twisted curve $E'(\mathbb{F}_{q^d})$. The mapping given above can be used to map Q' to the point Q on the curve E defined over \mathbb{F}_{q^k} , and Q can then be used as the image point in Miller's algorithm. Note that the x -coordinate of Q is defined over \mathbb{F}_{q^d} , and thus the denominator elimination technique applies. Barreto et al. [6, 7] also note that cryptographic operations which do not involve pairing computation, such as scalar multiplication, can be performed solely using arithmetic in \mathbb{F}_{q^d} . The points of $E'(\mathbb{F}_{q^d})$ can then be mapped back to $E(\mathbb{F}_{q^k})$ when needed for pairing computation. Scott [103] uses the twist idea to implement the Tate pairing efficiently using ordinary elliptic curves over \mathbb{F}_p with embedding degree $k = 2$.

The second technique given by Barreto et al. exploits the fact that the group $\psi(E'(\mathbb{F}_{q^d}))$ is a subgroup of the trace zero subgroup of $E(\mathbb{F}_{q^k})$. Therefore, an alternative to using twists is to simply choose a random $R \in E(\mathbb{F}_{q^k})$, and then set $Q = R - R^{q^d}$. Q is then a trace-zero point with the property that its x -coordinate is defined over \mathbb{F}_{q^d} . This method is especially useful for hyperelliptic curves of genus $g > 1$. However, the disadvantages are that generating random points over \mathbb{F}_{q^k} is slower than doing so over \mathbb{F}_{q^d} , and the ability to

speed up non-pairing based operations does not apply

3.3 Squared Pairings

Eisentrager et al [25] present algorithms to compute the squared Weil and Tate pairings on elliptic curves, and the squared Tate pairing on hyperelliptic curves. The squared Weil and Tate pairings are deterministic, unlike the plain Weil and Tate pairings (as originally defined). Furthermore, the authors claim a speedup of approximately 20% by computing the squared Tate pairing using their method over the plain Tate pairing, and by extension, the same speedup for computing the squared Weil pairing over the plain Weil pairing. The authors also present a method to compute the squared Tate pairing on hyperelliptic curves, and claim a speedup of approximately 30% on the standard algorithm. This algorithm is notable for being the first detailed implementation of a bilinear pairing on a hyperelliptic curve of genus 2.

However, there is no real advantage to computing the squared Weil or Tate pairings using the methods given by Eisentrager et al, as they have been surpassed by superior methods to compute the plain Weil or Tate pairings, as detailed in the previous section. Barreto et al show that there is no need to include random points in Miller's algorithm, as they can be defined over a subfield and are eliminated by the final exponentiation as a result. Therefore, there is no real advantage to the deterministic algorithms to compute the squared Weil and Tate pairings. The authors use Cantor's algorithm in the genus 2 case to double and add divisors and to extract the necessary functions required by Miller's algorithm. However, in practice Lange's explicit formulae (e.g. see [70]) for the group law would be used.

The relationship between the squared Weil/Tate pairings and the plain Weil/Tate pairings is investigated further by Kang and Park [58]. The authors show that a squared pairing can be transformed into a plain pairing when the image point is a trace zero point. As seen previously, for a random point $R \in E(\mathbb{F}_{q^k})$, a trace zero point Q can be generated

as $Q = R - R^{q^{k/2}}$ Let e_n be either the reduced Tate pairing or the Weil pairing, and let $P \in E(\mathbb{F}_q)$ Then Kang and Park show that

$$e_n(P, R)^{1-q^{k/2}} = e_n(P, R)^2 = e_n(P, Q),$$

as $1 - q^{k/2} \equiv 2 \pmod{n}$ This result shows that evaluating at a random point $R \in E(\mathbb{F}_{q^k})$ when computing a squared pairing is equivalent to evaluating at the trace zero point $Q = R - R^{q^{k/2}}$ when computing the corresponding plain pairing As the x -coordinate of a trace zero point is defined over a subfield, the denominator elimination technique applies Therefore, it may be concluded that there is no real advantage in computing squared pairings using the methods of Eisentrager et al [25], as it will always be more efficient to use a plain pairing with a trace zero point

In a separate paper, Eisentrager et al [24] present formulae which eliminate a field multiplication from the standard way of computing $[2]P + Q$, where P and Q are points on an elliptic curve This idea can be used to speed up both scalar multiplication and pairing computation Instead of constructing a function h_{2b+c} by first computing h_{2b} and then h_{2b+c} in an independent manner, the idea is to compute h_{2b+c} directly using the faster formulae However, this idea is not useful when the order has a low Hamming weight, as is normally the case It also does not take the standard denominator elimination idea into account Freeman [28] adapts this method to hyperelliptic curves of genus 2

3.4 Pairings on Hyperelliptic Curves

Duursma and Lee [23] were the first to examine pairing implementation on hyperelliptic curves in a constructive manner In particular, they introduce several optimisations to the computation of the Tate pairing on hyperelliptic curves of the form $C: y^2 = r^p - x + d$ over \mathbb{F}_{p^m} , where $d = \pm 1$, $p \equiv 3 \pmod{4}$ and $\gcd(m, 2p) = 1$ These curves have embedding degree $k = 2p$ Firstly, Duursma and Lee propose using a multiple of the group order of the form $p^{mm} + 1$ which has Hamming weight 2 in base p It is shown how the final addition

does not need to be evaluated, as the line function that is calculated is a vertical line function and is eliminated by the final exponentiation. Therefore, using the order $p^{2pm} + 1$ results in a loop of pm iterations (to the base p) with no logical decisions, which helps to simplify the implementation. An additional advantage is that the final exponentiation can be computed as

$$(p^{2pm} - 1)/(p^{pm} + 1) = (p^{pm} - 1)$$

As described previously, an exponentiation of this form can be trivially computed with a multiplication, inversion and some Frobenius actions in $\mathbb{F}_{p^{2pm}}$.

Duursma and Lee also propose computing pairings on hyperelliptic curves using points in the support as image elements rather than divisors. A reduced divisor on a genus g curve typically has g finite points in the support, i.e. $D = (P_1) + \dots + (P_g) - g(\infty)$. However, Duursma and Lee propose using degenerate divisors instead, which are divisors with support consisting of a single affine point, i.e. $D = (P) - (\infty)$. Rather than represent these divisors using Mumford notation, it is easier to simply work with the point P . Duursma and Lee also give an explicit formula to compute the line functions that are required in Miller's algorithm, rather than extract them from the addition process.

Lemma 2 *Let $C: y^2 = x^p - x + d$ be a hyperelliptic curve over \mathbb{F}_{p^m} , $d = \pm 1$ and $p \equiv 3 \pmod{4}$ and let $P = (x_P, y_P) \in C(\mathbb{F}_{p^m})$. Then the function*

$$h_P = y_P^p y - (x_P^p - x + d)^{(p+1)/2}$$

has divisor $(h_P) = p(P) + (-[p](P - \infty)) - p(\infty)$ where

$$-[p](P - \infty) = (x_P^{p^2} + d^p + d, y_P^{p^2}) - (\infty)$$

Combining this function with a loop size of pm iterations results in the following closed

formula to compute the Tate pairing

$$\langle P, \psi(Q) \rangle = f_P(\psi(Q)) = \prod_{i=1}^{pm} h_{[p^{i-1}]P}(\psi(Q))^{p^{pm-i}}$$

Using the double-and-add approach of Miller's algorithm, the accumulating variable $f \in \mathbb{F}_{p^{2mp}}$ is exponentiated to p each iteration of the loop. Although this exponentiation is efficient, it must be performed pm times in total, meaning that it has a non-trivial cumulative cost. However, Duursma and Lee show how it is possible to absorb the p^{pm-i} exponentiation into the formulae, thus eliminating the exponentiation to p from Miller's algorithm altogether. In addition, Duursma and Lee show how the loop size of pm iterations can be replaced with a loop of m iterations. Rather than loop to pm , the key idea is to loop to m and to absorb the power to p inside the explicit formulae.

Among the relevant hyperelliptic curves are two curves extremely suited to pairing implementation. These are the elliptic curves $E: y^2 = x^3 - x + d$ over \mathbb{F}_{3^m} , where $d = \pm 1$, as defined previously in Table 3.1. Note that the group order for these curves divides $p^{3m} + 1 = 3^{3m} + 1$. These curves have embedding degree $k = 6$, which is the maximum embedding degree for supersingular elliptic curves. Let $\rho \in \mathbb{F}_{3^3}$ be a root of $\rho^3 - \rho - d = 0$, and let $\sigma \in \mathbb{F}_{3^2}$ be a root of $\sigma^2 + 1 = 0$. Then the distortion map $\psi(x, y) = (\rho - x, \sigma y)$ supports the denominator elimination technique of Barreto et al. [5]. Scalar multiplication by 3 is extremely efficient on these curves, as exploited by Galbraith et al. [31]. Therefore, it is convenient to use a ternary basis in Miller's algorithm. The Duursma-Lee algorithm for computing the Tate pairing on these curves is given in Algorithm 4.

Kwon [66] adapts the techniques of Duursma and Lee to elliptic curves in characteristic 2. There are exactly three isomorphism classes of supersingular elliptic curves over \mathbb{F}_{2^m} , where m is odd, and Kwon's method applies to all such curves. Two curves in particular are suitable for pairing based cryptography, as they have the maximum embedding degree of $k = 4$ for supersingular elliptic curves in characteristic 2. These curves are defined as $E_d: y^2 + y = x^3 + x + d$, where $d \in \{0, 1\}$, as given previously in Table 3.1. These curves

Algorithm 4 The Duursma-Lee algorithm for the curve $E: y^2 = x^3 - x + d$ over \mathbb{F}_{3^m} , $d = \pm 1$

INPUT $P = (x_P, y_P), Q = (x_Q, y_Q) \in E(\mathbb{F}_{3^m})$

OUTPUT $\langle P, \psi(Q) \rangle$

```

1  $f \leftarrow 1$ 
2 for  $i \leftarrow 1$  to  $m$  do
3    $x_P \leftarrow x_P^3, y_P \leftarrow y_P^3$ 
4    $f \leftarrow f \cdot (-\sigma y_P y_Q - (x_P + x_Q - \rho + d)^2)$ 
5    $x_Q \leftarrow x_Q^{1/3}, y_Q \leftarrow y_Q^{1/3}$ 
6 end for
7 Return  $f$ 

```

support a doubling formula, such that for the point $P \in E(\mathbb{F}_{2^m})$, where $P = (x, y)$, $[2]P$ is given as

$$[2]P = (x^4 + 1, x^4 + y^4)$$

A distortion map for these curves is given in Table 3.2. Instead of using the group order to compute the Tate pairing, it is possible to use the multiple $2^{2m} + 1$ instead, as

$$2^{2m} + 1 = (2^m + 2^{(m+1)/2} + 1)(2^m - 2^{(m+1)/2} + 1)$$

As the final addition can be omitted, this results in a closed formula to compute the Tate pairing on these curves, with a loop size of $2m$ iterations. Following Duursma and Lee, Kwon shows how the loop to $2m$ can be reduced to m iterations, by absorbing the exponentiation to 2 into the formulae. Kwon shows that 7 multiplications in \mathbb{F}_{2^m} are required per iteration of the loop. This compares favourably with the characteristic 3 case as modified by Granger et al. [43], which costs 14 multiplications in \mathbb{F}_{3^m} per iteration. Kwon's algorithm is given in Algorithm 5. Note that the extension field $\mathbb{F}_{2^{4m}}$ is represented using the polynomial basis $\{1, x, x^2, x^3\}$, where $x^4 + x + 1 = 0$.

Kwon also gives a variant of the algorithm which requires no square rooting. Kwon's stated motivation is that square rooting in a finite field is an expensive operation, roughly equivalent to that of a multiplication with a precomputation. However, square rooting can

Algorithm 5 Kwon's algorithm for the curves $E: y^2 + y = x^3 + x + d$ over \mathbb{F}_{2^m} , $d = \{0, 1\}$

INPUT $P = (x_P, y_P), Q = (x_Q, y_Q) \in E(\mathbb{F}_{2^m})$

OUTPUT $\langle P, \psi(Q) \rangle$

```

1  $f \leftarrow 1$ 
2 for  $i \leftarrow 1$  to  $m$  do
3    $x_P \leftarrow x_P^2, y_P \leftarrow y_P^2$ 
4    $z \leftarrow x_P + x_Q, w \leftarrow z + x_P x_Q + y_P + y_Q + d$ 
5    $f \leftarrow f \cdot (w + z \cdot i + (z + 1) \cdot i^2)$ 
6    $x_Q \leftarrow x_Q^{1/2}, y_Q \leftarrow y_Q^{1/2}$ 
7 end for
8 Return  $f$ 

```

be as fast as squaring in characteristic 2 [27]. In characteristic 3, cube-rooting can also be performed efficiently (e.g. see Barreto [3]), although it is not as efficient as cubing. In this case, it is better to precompute all of the m cubes of a value $x_Q \in \mathbb{F}_{3^m}$ in a table, and to access the table in reverse order in the algorithm to obtain the cube-roots.

Choi and Lee [17] detail the efficient computation of the Tate pairing on hyperelliptic curves of genus 2 in a more general way than that of Duursma and Lee. Instead of using Cantor's algorithm to derive the functions that are required in Miller's algorithm, Choi and Lee use Lange's explicit formulae [70] for the group law on genus 2 curves. These formulae are modified slightly to obtain the required functions, as the formulae are designed for scalar multiplication and hence do not calculate the line function that is required in an explicit manner. The formulae given by Choi and Lee are actually identical to formulae given in an earlier (Japanese only) paper by Takahashi [111]. Choi and Lee then present the first computational results for the Tate pairing on a hyperelliptic curve of genus $g > 1$.

The maximum embedding degree of a supersingular genus 2 curve over a large prime field \mathbb{F}_p is $k = 6$. However, no example of such curves is known. Instead, Choi and Lee implement the Tate pairing on the curve $y^2 = x^5 + a$, $a \in \mathbb{F}_p^*$, where $p \equiv 2, 3 \pmod{5}$. This is a supersingular genus 2 curve with embedding degree $k = 4$. This curve has a distortion map $\psi(x, y) \mapsto (\zeta_5 x, y)$ where ζ_5 is a primitive 5-th root of unity in \mathbb{F}_{p^4} . Note that this distortion map does not support the denominator elimination technique. The group order of this curve is $\#\text{Pic}_C^0(\mathbb{F}_p) = p^2 + 1$. Choi and Lee choose $\log_2(p) \approx 256$ and choose

a prime factor n of $\#P_{1C}^0(\mathbb{F}_p)$ such that $\log_2(n) \approx 160$. On a 2 GHz Pentium IV their timings to compute the Tate pairing vary between 515 and 594 ms, depending on the form of the divisors and whether precomputation is used or not.

Harasawa et al [45] construct a distortion map for the genus 2 curve $C: y^2 = x^5 - \alpha x$ over \mathbb{F}_{5^m} , where $\alpha = \pm 2$. Secondly, they show how to compute the modified Tate pairing on this curve. The authors take advantage of a simple quintuple operation for computing $[5]P$ for a point $P \in C(\mathbb{F}_{5^m})$, by rewriting Miller's algorithm to the base 5. The authors compare the efficiency of their algorithm to that of the prime field curve utilised by Choe and Lee [17], as both curves have embedding degree $k = 4$. Harasawa et al give a theoretical metric to show that their method is about twice as efficient as the metric given by Choe and Lee. However, this claim does not take into account the fact that it is easier to optimise arithmetic in \mathbb{F}_p than in \mathbb{F}_{5^m} .

3.5 Compressed Pairings

Scott and Barreto [106] show how to compress pairing values and how to speed up the subsequent exponentiation of these elements. The first contribution of this paper uses Lucas sequences to speed up the final exponentiation required to compute the Tate pairing. Lucas sequences provide an efficient means of implementing exponentiation in a subgroup of $\mathbb{F}_{q^k}^*$ whose order divides $q^{k/2} + 1$. An efficient laddering algorithm has been developed (e.g. see Joye and Quisquater [54]) to compute Lucas sequences. The laddering algorithm requires very little memory, facilitates parallel computing, and has a natural resistance to side-channel attacks [55].

Lucas sequences consist of a pair of functions $U_n(a, b)$ and $V_n(a, b)$, evaluating as elements of $\mathbb{F}_{q^{k/2}}$. Let $b = 1$, in which case the arguments to U_n and V_n can be omitted. The sequences are given as

$$\begin{aligned} U_0 &= 0, U_1 = 1, U_{n+1} = aU_n - U_{n-1}, \\ V_0 &= 2, V_1 = a, V_{n+1} = aV_n - V_{n-1} \end{aligned}$$

The laddering algorithm to compute V_n is given in Algorithm 6. It is only necessary to compute V_n , as U_n can be computed from V_n with the formula

$$U_n = \frac{aV_n - 2V_{n-1}}{a^2 - 4}$$

Algorithm 6 Computing Lucas sequence elements

INPUT $a, n_t = (n_{t-1} \ n_0)_2$, with $n_{t-1} = 1$

OUTPUT $V_n = V_n(a, 1)$

```

1  $v_0 \leftarrow 2$ 
2  $v_1 \leftarrow a$ 
3 for  $i \leftarrow t - 2$  downto 0 do
4   if  $n_i = 1$  then
5      $v_0 \leftarrow v_0 v_1 - a$ 
6      $v_1 \leftarrow v_1^2 - 2$ 
7   else
8      $v_1 \leftarrow v_0 v_1 - a$ 
9      $v_0 \leftarrow v_0^2 - 2$ 
10  end if
11 end for
12 Return  $v_0$ 

```

The final exponentiation required to compute the Tate pairing is $(q^k - 1)/n$. Assuming that the embedding degree k is even, an element $r \in \mathbb{F}_{q^k}$ can be represented using a polynomial basis as $x = (a + b\sqrt{\beta})$, where $a, b \in \mathbb{F}_{q^{k/2}}$ and $r^2 - \beta = 0$ is an irreducible polynomial over $\mathbb{F}_{q^{k/2}}$. The conjugate of r with respect to $\mathbb{F}_{q^{k/2}}$ can be computed as $\bar{r} = (a - b\sqrt{\beta})$. As the embedding degree k is even, $(q^{k/2} - 1)$ can be factored out of the final exponentiation, and can be trivially evaluated. Sometimes, other easily computed factors may also be extracted, but an expensive exponentiation to $\phi_k(q)/n$ remains, where ϕ_k is the k -th cyclotomic polynomial.

After exponentiating to $(q^{k/2} - 1)$, the element $r \in \mathbb{F}_{q^k}$ will have norm 1. In other words, the product of x by its conjugate with respect to $\mathbb{F}_{q^{k/2}}$ will be equal to 1, i.e.

$$r\bar{r} = (a + b\sqrt{\beta})(a - b\sqrt{\beta}) = a^2 - b^2\beta = 1$$

Note that an element of norm 1 can be determined up to the sign of b from a alone. Therefore, the output of the Tate pairing can be represented as one element in $\mathbb{F}_{q^{k/2}}$ and a single bit to determine the sign of b , rather than the full value in \mathbb{F}_{q^k} , thus giving a compression factor of (almost) 2. An element of norm 1 also has the property that an otherwise expensive field inversion can be computed with a simple conjugation. This follows from $\iota\bar{\iota} = 1$ and therefore $1/\iota = \bar{\iota}$.

Scott and Barreto show how to efficiently raise an element $x \in \mathbb{F}_{q^k}$ of norm 1 to the power m by means of Lucas sequences with the formula

$$x^m = (a + b\sqrt{\beta})^m = \frac{V_m(2a)}{2} + U_m(2a)b\sqrt{\beta}$$

As stated previously, only $V_m(2a)/2$ needs to be explicitly calculated. Scott and Barreto propose using this formula to compute the expensive exponentiation to $m = \phi_k(q)/n$ required for the Tate pairing, where k is even. The cost to compute ι^m using the laddering algorithm given in Algorithm 6 is $\bar{M} + \bar{S}$ per iteration, where \bar{M} and \bar{S} are a multiplication and a squaring respectively in $\mathbb{F}_{q^{k/2}}$. The conventional binary algorithm takes around $S + M/2$ per iteration assuming a random exponent, where S and M are a squaring and a multiplication respectively in \mathbb{F}_{q^k} . This is roughly equivalent to $3\bar{S} + 3\bar{M}/2$, assuming the ratios $3\bar{M} \sim M$ and $3\bar{S} \sim S$. Therefore, the Lucas sequence approach gives a speedup of about 60% over the basic binary square-and-multiply algorithm.

The second contribution of Scott and Barreto is to show how to compress pairing values to half length assuming an even embedding degree. For an element $\iota \in \mathbb{F}_{q^k}$, the trace with respect to $\mathbb{F}_{q^{k/2}}$ is equal to $\text{Tr}_{\mathbb{F}_{q^k}/\mathbb{F}_{q^{k/2}}}(x) = x + x^{q^{k/2}}$. Let $x = (a + b\sqrt{\beta}) \in \mathbb{F}_{q^k}$ be the output of the Tate pairing after the final exponentiation. Then

$$\text{Tr}_{\mathbb{F}_{q^k}/\mathbb{F}_{q^{k/2}}}(\iota) = \iota + \iota^{q^{k/2}} = (a + b\sqrt{\beta}) + (a - b\sqrt{\beta}) = 2a$$

As the second component of ι has been discarded, the pairing has been compressed to half length. This idea can be effectively combined with that of using Lucas sequences to

compute the final exponentiation. Scott and Barreto also show how to compress pairings to a third of their length when the curve has an embedding degree that is a multiple of 6. This involves using the trace with respect to $\mathbb{F}_{q^6/\mathbb{F}_3}$. Note that any subsequent exponentiation of compressed pairing values must take into account the fact that they are traces of full pairing values, they cannot be handled as general finite field elements.

Granger et al. [43] adopt techniques from torus-based cryptography to achieve pairing compression. First of all, Granger et al. give an alternative to computing the final exponentiation required to compute the reduced Tate pairing for the supersingular elliptic curves in characteristic 3 as studied by Duursma and Lee [23]. These curves have an embedding degree of $k = 6$. Duursma and Lee propose computing the Tate pairing on these curves using the order $q^3 + 1 = 3^{3m} + 1$. The output of Miller's algorithm prior to the final exponentiation is then an element of the quotient group

$$G = \mathbb{F}_{q^6}^* / (\mathbb{F}_{q^6}^*)^{q^3+1}$$

Exponentiating an element in $\mathbb{F}_{q^6}^*$ to $q^3 + 1$ gives an element in $\mathbb{F}_{q^3}^*$, as this exponentiation is the norm map with respect to \mathbb{F}_{q^6} . Therefore G simplifies to $\mathbb{F}_{q^6}^* / \mathbb{F}_{q^3}^*$. Two elements $a, b \in \mathbb{F}_{q^6}^*$ are equivalent (when considered as elements of G) if $a = b\epsilon$, where $\epsilon \in \mathbb{F}_{q^3}^*$. Let $a \in \mathbb{F}_{q^6}^* / \mathbb{F}_{q^3}^*$ be the output of the Tate pairing prior to the final exponentiation. Then exponentiating to the power of $(q^3 - 1)$ yields a unique value suitable for cryptographic purposes, as $a^{(q^3-1)} = (b\epsilon)^{(q^3-1)} = b^{(q^3-1)}$.

Let elements of the field \mathbb{F}_{q^6} be represented using a polynomial basis as $a = (a_0 + a_1\sqrt{\beta}) \in \mathbb{F}_{q^6}$, where $a_0, a_1 \in \mathbb{F}_{q^3}$, and β is a suitable quadratic non-residue. Then the output of Miller's algorithm $a \in \mathbb{F}_{q^6}^* / \mathbb{F}_{q^3}^*$ is written as

$$a = b\epsilon = (cb_0 + cb_1\sqrt{\beta})$$

Note that dividing by cb_1 gives the value $a' = b' = b_0/b_1 + \sqrt{\beta}$. As the value c has been eliminated, a' can be used as a unique representative of the coset of G to which a belongs.

Therefore, the final exponentiation is just a multiplication and inversion in \mathbb{F}_{q^3} . This is in contrast to the standard exponentiation to $q^3 - 1$, which requires a multiplication and an inversion in \mathbb{F}_{q^6} . However, the alternative means of securing a unique value is essentially no more efficient than the standard approach, as both inversion and multiplication in \mathbb{F}_{q^6} can be efficiently reduced to their counterparts in \mathbb{F}_{q^3} .

The approach of Granger et al. has two interesting implications. The first is that it results in a two-fold compression of the pairing value. The output of the pairing $a' = a_0/a_1 + \sqrt{\beta}$ can be represented by the element $a_0/a_1 \in \mathbb{F}_{q^3}$, which results in a two-fold compression. It is important to note that this value cannot be treated simply as a general element of \mathbb{F}_{q^3} . The second advantage is that any subsequent exponentiation of the pairing value is faster than a general exponentiation in \mathbb{F}_{q^6} . Each time a multiplication must be performed in the square-and-multiply algorithm to compute the exponentiation, the accumulating value is multiplied by the value $(a_0/a_1 + \sqrt{\beta})$. Writing $x = a_0/a_1$, observe that

$$(x + \sqrt{\beta})(b_0 + b_1\sqrt{\beta}) = (xb_0 - b_1) + (xb_1 + b_0)\sqrt{\beta}$$

The multiplication of two generic elements in \mathbb{F}_{q^6} costs 3 multiplications in \mathbb{F}_{q^3} using the Karatsuba approach. However, this method costs only 2 multiplications in \mathbb{F}_{q^3} .

Granger et al. then remark that the output of the Tate pairing on an elliptic curve over \mathbb{F}_q may be viewed as an element of an algebraic torus. Rubin and Silverberg [97] introduce the concept of torus-based cryptography as an alternative to using traces to obtain compression. Granger et al. give an alternative representation for the quotient group G as $G = T_2(\mathbb{F}_{q^3})$. This enables compression by a factor of 2. Additionally, Granger et al. show how the torus $T_6(\mathbb{F}_q)$ gives a compression ratio of 3 for these curves.

Granger et al. also show how to use loop-unrolling to speed up the algorithm given by Duursma and Lee [23] to compute the Tate pairing for certain supersingular elliptic curves of characteristic 3. Some of the terms in the representation of the line function

that is calculated at each iteration of the loop are equal to zero. This can be exploited by unrolling the loop times two, and by writing a special multiplication routine to multiply the two sparse functions together, before multiplying the result with the accumulating variable. This approach costs only 14 multiplications in \mathbb{F}_{3^m} per iteration of the loop. This compares favourably the original Duursma-Lee algorithm (20 multiplications) and the trace-based variant by Scott and Barreto [106] (17 multiplications).

3.6 The Weil Pairing

Koblitz and Menezes [65] examine the task of computing the Weil pairing on elliptic curves in detail. In order to compare the performance of the Tate pairing and the Weil pairing in a fair manner, Koblitz and Menezes give a metric for estimating the running time of pairings. Firstly, the cost of arithmetic in the finite field \mathbb{F}_{q^k} is analysed, by introducing so-called pairing-friendly fields. For the rest of this section, let $k \geq 2$ be even and $q = p$. Let s and m be a squaring and a multiplication respectively in \mathbb{F}_q . Similarly, let S and M be a squaring and a multiplication respectively in \mathbb{F}_{q^k} , and \bar{S} and \bar{M} be a squaring and multiplication in $\mathbb{F}_{q^{k/2}}$. Assume that $S \approx M$, $s \approx m$ and $\bar{S} \approx \bar{M}$. Also note that multiplying an element in \mathbb{F}_{q^k} by an element in \mathbb{F}_q takes time km .

Let $k = 2^i 3^j$ and $q \equiv 1 \pmod{12}$, and let $\beta \in \mathbb{F}_q$ be neither a square nor a cube in \mathbb{F}_q . Then the binomial $x^k - \beta$ is irreducible over \mathbb{F}_q and hence defines the extension field \mathbb{F}_{q^k} . Therefore, \mathbb{F}_{q^k} can be constructed from \mathbb{F}_q as a tower of quadratic and cubic extensions, by adjoining the squareroot or cuberoot of β , then the squareroot or cuberoot of that, etc. Using the Karatsuba [59] technique, a multiplication in a quadratic extension takes 3 multiplications in the subfield. This technique will be detailed later in this thesis. Similarly a multiplication in a cubic extension takes 5 multiplications in the subfield, using the Toom-Cook [112, 20] method. However, this analysis omits the large amount of additions and divisions by constants that must be performed with this method. Using these two multiplication methods, Koblitz and Menezes estimate the cost of a multiplication in \mathbb{F}_{q^k} as

$$M \approx 3.5' m$$

As discussed before, it is common that the subgroup order n is defined to have a low Hamming weight. In this case, the number of additions in Miller's algorithm is negligible compared to the number of doublings. Therefore, Koblitz and Menezes analyse the cost of computing a pairing by focusing solely on the computation that takes place when a point doubling is performed. To compute the Weil pairing $e_n(P, Q)$, the functions l_1/v_1 and l_2/v_2 are extracted from the addition process for each bit of n . These functions can be accumulated by using the two variable algorithm of Galbraith et al. [31] in the following way

$$\frac{f_1}{f_2} = \frac{f_1^2 v_2(Q) l_1(Q)}{f_2^2 l_2(Q) v_1(Q)}$$

This function, along with the associated point doubling, is termed a Miller operation. Koblitz and Menezes point out that the denominator elimination technique also applies to the Weil pairing assuming that the output of the pairing is exponentiated to $(q^{k/2} - 1)$ (alternatively to $(1 - q^{k/2})$), which can be trivially realised as a conjugation with respect to $\mathbb{F}_{q^{k/2}}$, and a multiplication and an inversion in \mathbb{F}_{q^k} . In this case, the function evaluation simplifies to

$$\frac{f_1}{f_2} = \frac{f_1^2 l_1(Q)}{f_2^2 l_2(Q)}$$

Note that for the Tate pairing this simplifies to $f_1 = f_1^2 l_1(Q)$

As the Weil pairing consists of two loops, the first is termed Miller lite (due to Solinas [110]), as the iterating point is defined over \mathbb{F}_q . The second loop is called the full Miller loop, as the iterating point is defined over \mathbb{F}_{q^k} . The full Miller loop is much more computationally expensive than the Miller lite loop, as the arithmetic is in \mathbb{F}_{q^k} . However, Koblitz and Menezes follow the idea of Barreto et al. [6, 7], in defining the image point Q as a point on a quadratic twist of the curve defined over the quadratic subfield $\mathbb{F}_{q^{k/2}}$. This leads to a gain both in evaluating at Q in the Miller lite loop, and in performing arithmetic on Q in

Table 3 3 Minimum bitlengths of n and q^k

Security level (in bits)	80	128	192	256
b_n	160	256	384	512
b_{q^k}	1024	3072	8192	15360
$\gamma = b_{q^k}/b_n$	6 4	12	21 33	30

Table 3 4 Operation counts for each bit of n

k	Final exponentiation	Miller lite	Full Miller
$k = 2$	$(\gamma/2 - 1)(s + m)$	$4s + 8m + S + M$	$4s + 8m + S + M$
$k \geq 4$	$(\tau_k \gamma - 1)(\bar{S} + \bar{M})$	$4s + (k + 7)m + S + M$	$km + 4\bar{S} + 6\bar{M} + S + M$

the full Miller loop To compare the Tate pairing with the Weil pairing, the cost of the full Miller loop must be compared with the final exponentiation to $(q^{k/2} + 1)/n$ required by the Tate pairing (as both pairings have the $(q^{k/2} - 1)$ exponentiation in common)

Koblitz and Menezes estimate the cost of the Miller operation for the Miller lite loop as $4s + 8m + S + M$ for $k = 2$, and $4s + (k + 7)m + S + M$ for $k \geq 4$ assuming the use of Jacobian coordinates The cost of the full Miller loop is the same as Miller lite for $k = 2$, and is given as $km + 4\bar{S} + 6\bar{M} + S + M$ for $k \geq 4$ Koblitz and Menezes use the security parameters defined by Lenstra [73], that are reproduced in Table 3 3 Let b_n be the number of bits of the prime subgroup order n , b_{q^k} be the number of bits of q^k and let $\gamma = b_{q^k}/b_n$ Let $\tau_k = 1/2$ if $k = 2^i, i \geq 1$, else let $\tau_k = 1/3$ (if $k = 2^i 3^j, i, j \geq 1$) Then, using the Lucas sequence approach of Scott and Barreto [106], Koblitz and Menezes estimate the cost of the exponentiation to $(q^{k/2} + 1)/n$ as $(\tau_k \gamma - 1)(\bar{S} + \bar{M})$ for each bit of n These results are summarised in Table 3 4

For the embedding degree $k = 2$, Koblitz and Menezes estimate that the Tate pairing will be faster when $\gamma < 20$ However, they estimate that the Weil pairing will become more efficient to compute for higher values of γ , starting at the 192-bit security level When $k \geq 4$, the Weil pairing becomes more efficient to compute when $\gamma = 28 8$ for $k = 6$, $\gamma = 28 2$ for $k = 12$ and $\gamma = 27 8$ for $k = 24$ Therefore, when $k \geq 4$ the Weil pairing becomes more efficient than the Tate pairing at the 256-bit security level

Granger et al [42] examine pairing implementation using ordinary elliptic curves and various practical levels of security. Firstly, they analyse the cost of computing the final exponentiation required for the Tate pairing. Recall, that for an even embedding degree, the factor $(q^{k/2} - 1)$ can be extracted from the final exponentiation and easily evaluated. The remaining exponentiation that must be performed can sometimes be simplified further as $(q^{k/2} + 1)/\phi_k(q)$ and $\phi_k(q)/n$. However, the exponentiation to $\phi_k(q)/n$ is an expensive operation. Rather than use the Lucas sequence approach of Scott and Barreto, Hu et al [50] introduce the idea of exploiting the q -th power Frobenius endomorphism to compute this exponentiation. This can be done by simply writing $\phi_k(q)/n$ to the base q . Granger et al suggest using multi-exponentiation (e.g. see Avanzi [2]) to compute all of the resulting exponentiations using a single square-and-multiply algorithm.

Granger et al examine the theoretical costs of using this method to compute the final exponentiation. They conclude that the Lucas sequence approach is more efficient for embedding degree $k \leq 6$. However, for $k > 6$ the multi-exponentiation idea is more efficient. As discussed previously, Koblitz and Menezes conclude that the Weil pairing is more efficient than the Tate pairing at high levels of security. However, this analysis does not take into account the technique of multi-exponentiation. Granger et al conclude that the Tate pairing is always faster than the Weil pairing for all of the interesting security sizes used in practice.

Scott [105] also examines the relative efficiency of the Tate pairing and the Weil pairing. In public key schemes that are based on the DLP in $\mathbb{F}_{q^k}^*$, security is traditionally increased by increasing the size of q , for example from 1024 bits to 2048 bits. However, this leads to a substantial increase in the cost of arithmetic in $\mathbb{F}_{q^k}^*$, which can be problematic in constrained environments. Scott points out that pairing based cryptography has another option, to keep the size of the underlying field constant and to double the embedding degree k . This has the added advantage of requiring only minimal changes to the underlying software or hardware implementation. Scott advocates fixing the base field size at 512 bits, and using elliptic curves with embedding degrees 2, 4 and 8, depending on the level of security that is

required

Scott also addresses the practical considerations of computing the expensive $\phi_k(q)/n$ exponentiation required for the Tate pairing. Using the techniques of Granger et al [42], Scott points out that it is easiest to precompute $\phi_k(q)/n$ and store it as a number to the base q , and then to exploit the q -th power Frobenius action to allow a multi-exponentiation. Scott also shows how the entire final exponentiation to $(q^k - 1)/n$ can be included in the multi-exponentiation. This approach does not require an inversion, which may be useful in restricted environments where inversion is particularly expensive. Scott provides experimental evidence that the exponentiation to $\phi_k(q)/n$ is faster using the Lucas sequence approach when $k \leq 4$.

Scott then gives an algorithm to compute the Weil pairing for an even embedding degree, which is more efficient than that given by Koblitz and Menezes. The algorithm given by Koblitz and Menezes includes an exponentiation to $(q^{k/2} - 1)$ after the loop, to avoid computing the vertical line functions. However, raising the output of Miller's algorithm to this power also means that inversion can be replaced inside the loop with a simple conjugation with respect to $\mathbb{F}_{q^{k/2}}$. Thus Scott shows that only one accumulating variable is needed to compute the Weil pairing, rather than the two variable approach previously used in analysing the cost of computing the Weil pairing.

It is possible to precompute the required multiples of the first input point to the Tate pairing and to store them for use in Miller's algorithm. This technique is useful if a pairing is computed multiple times using the same iterating point, and if storage space is not an issue. Scott shows that this technique is applicable to the second input point when computing the Weil pairing. This greatly reduces the computational complexity of the Weil pairing, as it is no longer required to double and add the point defined over the extension field inside the algorithm. Scott concludes by giving experimental results that validate the assertion that the Tate pairing is faster than the Weil pairing for all interesting security levels, except when it is possible to precompute points.

The results mentioned in this section so far are applicable to all curves, as no special

properties are exploited. However, Park et al [88] describe a technique to implement the Weil pairing efficiently on supersingular curves, when the modified pairing is used. In particular, they show how the full Miller loop can be effectively replaced with a Miller lite loop. First of all, the distortion map that is used must be separable.

Definition 26 *An endomorphism ϕ is inseparable if and only if*

$$\phi(x, y) = (u(x^p, y^p), v(x^p, y^p))$$

for some rational functions u, v where p is the characteristic of \mathbb{F}_q .

Most distortion maps used in practice are of degree one and are therefore separable. Park et al then show how the Weil pairing can be computed as

$$e_n(P, \psi(Q)) = \frac{f_P(\psi(Q) + R)}{f_P(R)} \frac{f_Q(-\psi^{-1}(R))}{f_Q(\psi^{-1}(P - R))}$$

In fact, the random point R can be omitted as explained earlier. Clearly, this definition replaces the full Miller loop on the point $\psi(Q)$ with a Miller lite loop on Q . Also, the function evaluation is at $\psi^{-1}(P)$ rather than at P , where ψ^{-1} is the inverse of the distortion map. Park et al then show that a self-pairing can be computed as

$$e_n(P, \psi(P)) = c_\psi^n \frac{f_P(\psi(P))}{f_P(\psi^{-1}(P))},$$

where c_ψ^n is a constant that depends on the distortion map. Self-pairings are required for a certain number of cryptographic applications. Park et al's self-pairing formula can be computed with a single Miller loop, and it has no final exponentiation. However, the complexity of the algorithm is not the same as that of the Tate pairing without the final powering, contrary to the claim of Park et al. This is due to the fact that it is necessary to evaluate the line functions at $\psi(P)$ and $\psi^{-1}(P)$ each iteration of the loop, whereas only one evaluation is required using the standard denominator elimination technique with the Tate pairing.

3.7 More Recent Optimisations

Hu et al [50] implement the Tate pairing using a supersingular elliptic curve over \mathbb{F}_{p^2} with an embedding degree of $k = 3$. This is the first reported implementation in the literature of a pairing on a curve with an odd embedding degree. As detailed earlier, this paper also shows how the Frobenius endomorphism can be exploited to speed up the final exponentiation required for the Tate pairing. The disadvantage to using an odd embedding degree is that the denominator elimination technique of Barreto et al [5] does not apply. Therefore, Hu et al [50] use the algorithm of Galbraith et al [31]. Another argument against using this curve is that the iterating point is defined over \mathbb{F}_{p^2} , instead of \mathbb{F}_p as is more common.

Blake et al [8] give some refinements to Miller's algorithm for general elliptic curves. Their improvements reduce the total number of line functions in Miller's algorithm. However, as these techniques do not incorporate denominator elimination, they are not particularly useful in practice. Let $h(Q)$ be a linear function in two variables that is evaluated at the point $Q = (x, y)$. Then the conjugate of $h(Q)$, which is denoted $\overline{h(Q)}$, is equal to $h(-Q)$, where $-Q$ is the opposite of Q . Let $l_P(Q)$ be the evaluation of the point Q at the line function when doubling P , and let $v_{[2]P}(Q)$ be the vertical line through $[2]P$. Then

$$-l_P(Q)l_P(-Q) = -l_P(Q)\overline{l_P(Q)} = v_P^2(Q)v_{[2]P}(Q)$$

The minus sign can be omitted in Miller's algorithm, as the pairing value is not affected by non-zero constants. This technique is used later in this thesis to prove a result about the Tate pairing.

Scott [104] shows how to efficiently implement the Tate pairing on certain ordinary elliptic curves. These curves are closely related to the supersingular elliptic curves used by Boneh and Franklin [11]. However, Scott shows that a speedup of up to 20% is possible when computing the Tate pairing in the ordinary elliptic curve case. Alternatively, only half the amount of storage is required if it is possible to precompute the lines that are required in Miller's algorithm. The ordinary curves in question have the same equations as the two

Boneh and Franklin supersingular elliptic curves, but with different congruence conditions attaching to the large prime μ . So the curves are no longer supersingular and have no distortion maps as a result.

However, the same maps are still endomorphisms of the curve and are useful in the context of scalar multiplication, as examined by Gallant et al. [37]. Gallant et al. give endomorphisms for these curves such that given a point P , a fixed multiple of the point can be determined with a single field multiplication. Scott transfers this idea to the area of pairing computation, by observing that the line functions in the first half of Miller's algorithm are related by the endomorphism to the line functions in the second half. Therefore, the line functions from the first half of the algorithm can be stored, before they are multiplied by the accumulating variable. In the second half of the loop, the group arithmetic can be avoided by reusing the stored functions.

3.8 Conclusion

Various methods in the literature to compute pairings efficiently have been described in this chapter. A number of conclusions arise naturally from the optimisations that have been detailed. Firstly, a number of generic techniques exist that improve the running time of Miller's algorithm as originally defined. The first input divisor should be defined over the field \mathbb{F}_q , rather than \mathbb{F}_{q^k} . In this way, the group arithmetic takes place over the smaller field, which is a large saving. An even embedding degree should be used with a distortion map for supersingular curves, in order to use the denominator elimination technique. Any random divisor used to guarantee the non-degeneracy of the Tate pairing can be omitted as it can be defined over a subfield, and hence eliminated by the final exponentiation.

Secondly, the Tate pairing should be used, rather than the Weil pairing, as it is always more efficient to compute. The final exponentiation can be evaluated reasonably efficiently using either Lucas sequences or multi-exponentiation, depending on the embedding degree of the curve in question. Thirdly, numerous optimisations exist when computing the Tate

pairing using certain supersingular curves over low characteristic. These optimisations include a shortened loop size, a trivial final exponentiation, and no conditional statements in the loop.

There is no inherent obstacle to using any of these optimisations to compute pairings efficiently on genus 2 curves. However, the papers that have been described in this chapter are largely unclear on this issue. Duursma and Lee [23] provide a family of hyperelliptic curves suitable for fast pairing implementation. However, this family of curves contains no hyperelliptic curves of genus 2 over finite fields of a suitable characteristic. Any implementation of genus 2 pairings that exists in the literature lies far behind the equivalent implementation on elliptic curves. This deficit is addressed in the remainder of this thesis.

Chapter 4

Pairings on Supersingular Genus 2

Curves over \mathbb{F}_{2^m}

4.1 Introduction

In this chapter, the first efficient implementation of the Tate pairing on a supersingular genus 2 curve over \mathbb{F}_{2^m} is described. Firstly, various supersingular curves over \mathbb{F}_{2^m} are examined, and two curves are selected that have the maximum embedding degree of $k = 12$ for genus 2 curves in characteristic 2. It is shown how to compute the group order for these curves, and how to select the field \mathbb{F}_{2^m} such that the group order has a large prime factor. Explicit formulae are given for doubling divisors in $\text{Pic}_C^0(\mathbb{F}_{2^m})$ for the curves in question.

Various aspects of the arithmetic of the selected curves are then explored. It is shown how to construct the extension fields that are required, and how to perform arithmetic in these fields in an efficient manner. An octupling automorphism is given on the curves, which can be exploited in Miller's algorithm by using an octic basis. Degenerate divisors and their application to pairing computation are examined. Explicit formulae are derived for the intermediate functions that are required in Miller's algorithm, and it is shown how the final exponentiation can be computed efficiently.

The implementation of the Tate pairing is then considered in detail using an octic ba-

sis and explicit formulae. As degenerate divisors are used, the input elements to Miller's algorithm are points on the curve. However, using a standard "double-and-add" algorithm destroys the special form of the iterating divisor, due to the additions in the group order. It is shown how this can be avoided by splitting the function that is required into several other functions which are computed separately. It is also shown how a large amount of computation in Miller's algorithm can be avoided by precomputing certain powers of the first input point.

Whenever a doubling is performed in Miller's algorithm, the accumulating variable must be squared. As an octic basis is used in this chapter, this operation must be performed 3 times per iteration of the loop. However, it is shown how this can be avoided by building the exponentiation into the explicit formulae inside the loop, at the cost of some extra operations in the extension field. This optimisation requires the precomputation of certain powers of the second input point. Finally, some experimental results are given and the chapter is concluded.

This chapter contains joint work with Paulo S L M Barreto, Steven D Galbraith and Michael Scott, which has been accepted for publication in *Designs, Codes and Cryptography*. A preprint is available at the ePrint archive as Barreto et al [4].

4.2 The Curve

The first task is to select a suitable genus 2 curve over \mathbb{F}_{2^m} with a low embedding degree. As no ordinary genus 2 curves are known that have a low embedding degree, the search must be restricted to supersingular curves. In the context of finite fields of characteristic 2, Koblitz curves are curves that are defined over the binary field \mathbb{F}_2 , and the degree zero divisor class group of the curve is considered over \mathbb{F}_{2^m} , for some prime m . As detailed in Chapter 2, it is a simple matter to compute the group order $\#\text{Pic}_C^0(\mathbb{F}_{2^m})$ for Koblitz curves. Also, as the coefficients of the equation of the curve are either 0 or 1, it is possible to reduce the computational complexity of the group law. Therefore, Koblitz curves are attractive for

pairing implementation For information on how to speed up scalar multiplication on genus 2 Koblitz curves see Gunther et al [44] and Lange [67]

Recall that a genus 2 curve is given by the equation $C: y^2 + h(x)y = f(x)$, where $f(x)$ is monic of degree 5, and $h(x)$ is of degree ≤ 2 Choie and Yun [18] classify genus 2 curves over \mathbb{F}_{2^m} into three types, which depend on the degree and form of the $h(x)$ polynomial Curves with a constant $h(x)$ are defined as Type-III curves The following lemma due to Galbraith [32] shows that all Type-III curves are supersingular Note that no genus 2 curve defined over \mathbb{F}_2 with a non-constant $h(x)$ polynomial is supersingular

Lemma 3 *Let C be a genus 2 curve over \mathbb{F}_{2^m} of the form $y^2 + cy = f(x)$ where $f(x)$ is monic of degree 5 and $c \in \mathbb{F}_{2^m}^*$. Then C is supersingular*

When considering the equation of Type-III curves over the field \mathbb{F}_2 , the left-hand side of the equation is fixed as $y^2 + y$. The right-hand side of the equation is $f(x) = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$, where all $f_i \in \{0, 1\}$. This implies that there are a maximum of 2^5 different curve equations for Type-III curves over \mathbb{F}_2 . However, by a linear change of variables, Choie and Yun show that all Type-III curves are of the form

$$C: y^2 + y = x^5 + f_3x^3 + f_1x + f_0$$

This equation implies a maximum of 2^3 possible curve equations. In fact, there are 6 different curves up to isomorphism, as the curve $y^2 + y = x^5 + 1$ is isomorphic to $y^2 + y = x^5$, and the curve $y^2 + y = x^5 + x^3 + x + 1$ is isomorphic to the curve $y^2 + y = x^5 + x^3 + x$. Therefore, there are essentially 6 different supersingular Koblitz curves of genus 2 over \mathbb{F}_{2^m} that must be investigated for pairing computation.

Table 4.1 gives a representative of each of the 6 different isomorphism classes of Type-III curves defined over \mathbb{F}_2 , along with the embedding degree of each curve. For all of these curves, Koblitz [64] gives an automorphism to compute a fixed scalar multiple $[2^e]P$ of a point $P \in C(\mathbb{F}_{2^m})$, by applying the 2nd power Frobenius endomorphism to the coordinates of P . The Frobenius endomorphism ϕ_2 is trivially computed in characteristic 2 as it equates

Table 4 1 Supersingular genus 2 curves over \mathbb{F}_2

Curve	Automorphism	Embedding degree
$y^2 + y = x^5 + x^3 + x$	$[8]P = \phi_{2^6}(P)$	3
$y^2 + y = x^5$	$[4]P = -\phi_{2^4}(P)$	4
$y^2 + y = x^5 + x$	$[16]P = \phi_{2^8}(P)$	4
$y^2 + y = x^5 + x + 1$	$[16]P = \phi_{2^8}(P)$	4
$y^2 + y = x^5 + x^3$	$[64]P = -\phi_{2^{12}}(P)$	12
$y^2 + y = x^5 + x^3 + 1$	$[64]P = -\phi_{2^{12}}(P)$	12

Table 4 2 Supersingular genus 2 curves over \mathbb{F}_2 with $k = 12$

Isomorphism class 1	Isomorphism class 2
$y^2 + y = x^5 + x^3$	$y^2 + y = x^5 + x^3 + 1$
$y^2 + y = x^5 + x^3 + x^2 + x$	$y^2 + y = x^5 + x^3 + x^2 + x + 1$
$y^2 + y = x^5 + x^4 + x^3 + x$	$y^2 + y = x^5 + x^4 + x^3 + x + 1$
$y^2 + y = x^5 + x^4 + x^3 + x^2$	$y^2 + y = x^5 + x^4 + x^3 + x^2 + 1$

to a squaring. As a result, the automorphisms that are given enable an extremely efficient method to perform the group operation. This is not useful for systems based on the DLP, as these curves are all supersingular and hence are vulnerable to the attack of MOV/FR. However, this property is exploited later in this chapter to compute the Tate pairing.

As can be seen in Table 4 1, two curves have the maximum embedding degree of $k = 12$ given by Rubin and Silverberg for genus 2 curves over \mathbb{F}_{2^m} . These curves are

$$C_d: y^2 + y = x^5 + x^3 + d, \quad d \in \{0, 1\}$$

Table 4 2 gives the other curve equations over \mathbb{F}_2 that are isomorphic to the curves C_d . However, as the curve equations C_d have the smallest number of coefficients of all of these curves, the other curves are not considered for pairing implementation in this chapter.

The next step is to determine the group order $\#\text{Pic}_C^0(\mathbb{F}_{2^m})$ for the curves C_d . Again, we largely follow Koblitz [64] in this treatment. The characteristic polynomial of the Frobenius

endomorphism for a genus 2 curve C over \mathbb{F}_2 is given as

$$\chi_C(T) = T^4 + a_1T^3 + a_2T^2 + 2a_1T + 4 = \prod_{i=1}^4 (T - \alpha_i),$$

where $a_1, a_2 \in \mathbb{Z}$, and the α_i are complex numbers of absolute value $\sqrt{2}$. As detailed in Chapter 2, once the coefficients a_1 and a_2 are known, the group order over \mathbb{F}_2 can be evaluated as $\chi_C(1) = \#\text{Pic}_C^0(\mathbb{F}_2)$. To determine the group order over some extension field \mathbb{F}_{2^m} , it is necessary to find the factors α_i of $\chi_C(T)$, and then to compute

$$\#\text{Pic}_C^0(\mathbb{F}_{2^m}) = \prod_{i=1}^4 (1 - \alpha_i^m)$$

To determine a_1 and a_2 for the curves C_d , it is necessary to first count the points on the curves over \mathbb{F}_2 and \mathbb{F}_{2^2} . The results are $\#C_0(\mathbb{F}_2) = 5$ and $\#C_0(\mathbb{F}_{2^2}) = 5$, and $\#C_1(\mathbb{F}_2) = 1$ and $\#C_1(\mathbb{F}_{2^2}) = 5$. The coefficients a_1 and a_2 of $\chi_C(T)$ are then computed as $a_1 = \#C(\mathbb{F}_2) - 3$ and $a_2 = (\#C(\mathbb{F}_{2^2}) - 5 + a_1^2)/2$. For C_0 these values are $a_1 = a_2 = 2$, and for C_1 these values are $a_1 = -2$, $a_2 = 2$. The characteristic polynomial of the Frobenius endomorphism for these curves is then

$$\chi_{C_d}(T) = T^4 + (-1)^d 2T^3 + 2T^2 + (-1)^d 4T + 4$$

It is now necessary to derive the α_i . First of all, the quadratic equation $x^2 + a_1x + (a_2 - 4) = 0$ must be solved to obtain the two roots γ_1 and γ_2 . For C_0 these roots are $-1 \pm \sqrt{3}$ and for C_1 the roots are $1 \pm \sqrt{3}$. The α_i are then found by solving the quadratic equation $x^2 - \gamma_i + 2 = 0$, and the group order over \mathbb{F}_{2^m} is computed as $\#\text{Pic}_C^0(\mathbb{F}_{2^m}) = \prod_{i=1}^4 (1 - \alpha_i^m)$. As m must be prime to avoid the Weil descent attack, the group order over \mathbb{F}_{2^m} can be written for C_0 as

$$\#\text{Pic}_{C_0}^0(\mathbb{F}_{2^m}) = 2^{2m} + (-1)^{\lfloor (m+1)/4 \rfloor} 2^{(3m+1)/2} + 2^m + (-1)^{\lfloor (m+1)/4 \rfloor} 2^{(m+1)/2} + 1,$$

and for C_1 as

$$\#\text{Pic}_{C_1}^0(\mathbb{F}_{2^m}) = 2^{2m} - (-1)^{\lfloor (m+1)/4 \rfloor} 2^{(3m+1)/2} + 2^m - (-1)^{\lfloor (m+1)/4 \rfloor} 2^{(m+1)/2} + 1$$

where $\lfloor \cdot \rfloor$ denotes the greatest integer function. When describing details of the pairing implementation later in this chapter, the group order for both curves is written for convenience as

$$\#\text{Pic}_C^0(\mathbb{F}_{2^m}) = 2^{2m} \pm 2^{(3m+1)/2} + 2^m \pm 2^{(m+1)/2} + 1$$

There are two criteria on the selection of the prime m . First of all, it should be large enough so that the group $\text{Pic}_C^0(\mathbb{F}_{2^m})$ is invulnerable to the Pollard-rho attack, and $\mathbb{F}_{2^{km}}^*$ is resistant to index calculus attacks. Secondly, the group order $\#\text{Pic}_C^0(\mathbb{F}_{2^m})$ must be divisible by a large prime number to avoid the attack of Pohlig and Hellman. In other words, only a small co-factor should divide the group order. As m is prime, only the group order over the base field, $\#\text{Pic}_C^0(\mathbb{F}_2)$, divides the group order over the full field, $\#\text{Pic}_C^0(\mathbb{F}_{2^m})$. Therefore, to give the maximum resistance to the attack of Pohlig and Hellman, $\#\text{Pic}_C^0(\mathbb{F}_{2^m})$ should be divisible by as small a multiple of $\#\text{Pic}_C^0(\mathbb{F}_2)$ as possible.

Computing the group order over \mathbb{F}_2 for both curves can be done by simply evaluating the characteristic polynomial of the Frobenius endomorphism at 1, which yields $\chi_{C_0}(1) = \#\text{Pic}_{C_0}^0(\mathbb{F}_2) = 13$ and $\chi_{C_1}(1) = \#\text{Pic}_{C_1}^0(\mathbb{F}_2) = 1$. Therefore, for the curve C_0 it is desirable to find a prime m , such that as small a multiple of 13 as possible divides $\#\text{Pic}_{C_0}^0(\mathbb{F}_{2^m})$, leaving a prime number. In fact, an exhaustive search yields some examples where the co-factor is the lowest possible value of 13. For C_1 , as $\#\text{Pic}_{C_1}^0(\mathbb{F}_2) = 1$, it is theoretically possible to find instances of m where the group order itself is prime. However, in the range of m which is large enough for security, yet small enough for practical implementation, only one such example was found. Table 4.3 gives a number of examples for both curves.

Lange [70] gives explicit formulae for performing the group arithmetic on genus 2

Table 4.3 \mathbb{F}_{2^m} , where $\#P_{10C}^0(\mathbb{F}_{2^m})$ is equal to a small cofactor times a prime

Finite Field	Curve	Co-factor
$\mathbb{F}_{2^{103}}$	$y^2 + y = x^5 + x^3$	13 1237
$\mathbb{F}_{2^{181}}$	$y^2 + y = x^5 + x^3$	13
$\mathbb{F}_{2^{211}}$	$y^2 + y = x^5 + x^3$	13
$\mathbb{F}_{2^{79}}$	$y^2 + y = x^5 + x^3 + 1$	151681
$\mathbb{F}_{2^{127}}$	$y^2 + y = x^5 + x^3 + 1$	198168459411337
$\mathbb{F}_{2^{199}}$	$y^2 + y = x^5 + x^3 + 1$	2389 121789
$\mathbb{F}_{2^{239}}$	$y^2 + y = x^5 + x^3 + 1$	1

curves over finite fields of arbitrary characteristic. These formulae are more efficient than the generic algorithm due to Cantor. As mentioned earlier, the group arithmetic for curves defined over \mathbb{F}_2 can be more efficient than for curves defined over \mathbb{F}_{2^m} , especially if the equation of the curve is sparse. In Algorithm 7, explicit formulae are given for doubling a divisor on the curves C_d . As this is the most common operation in scalar multiplication, the other cases can be handled by Cantor's algorithm. We note that formulae by Stevens given later in chapter 14 of Cohen et al. [19] slightly improve on the efficiency of some of these formulae.

4.3 Curve Arithmetic

In this section, all of the background information that is needed to implement pairings on the selected curves is described.

4.3.1 Finite field arithmetic

As the embedding degree of the curves C_d is $k = 12$, it is necessary to show how to construct the extension field $\mathbb{F}_{2^{12m}}$. A polynomial basis representation will be used rather than a normal basis representation, for reasons outlined in chapter 2. There are a number of different ways to construct the field $\mathbb{F}_{2^{12m}}$. As the curve is initially considered over the field \mathbb{F}_{2^m} , it makes sense to choose this field as the base field. Rather than construct $\mathbb{F}_{2^{12m}}$ as a degree 12 extension of \mathbb{F}_{2^m} , it is more convenient to first construct the field $\mathbb{F}_{2^{6m}}$, using

Algorithm 7 Doubling of a divisor $[u, v]$ on the curves C_d

INPUT $[u, v]$ OUTPUT $[u', v'] = 2[u, v]$

```
1 if  $\deg(u) = 0$  then
2    $[u', v'] \leftarrow [1 \ 0]$ 
3 else if  $\deg(u) = 1$  then
4    $u'_0 \leftarrow u_0^2, v'_1 \leftarrow u_0^2 + u'_0, v'_0 \leftarrow v_0^2 + d$ 
5    $[u', v'] \leftarrow [x^2 + u'_0, v'_1 x + v'_0] \triangleright (3S)$ 
6 else
7   if  $u_1 = 0$  then
8      $s'_0 \leftarrow v_1^2, l'_0 \leftarrow u_0 s'_0$ 
9      $u'_0 \leftarrow s_0^2$ 
10     $w_1 \leftarrow s'_0 + 1, w_0 \leftarrow w_1 + u'_0 + u_0, w_1 \leftarrow u'_0 w_1 + l'_0,$ 
11     $v'_1 \leftarrow w_0 + v_1, v'_0 \leftarrow w_1 + v_0 + 1$ 
12     $[u', v'] \leftarrow [x^2 + x + u'_0, v'_1 x + v'_0] \triangleright (2S, 2M)$ 
13  else if  $u_1 = 1$  then
14     $s'_0 \leftarrow v_1^2, u'_0 \leftarrow s_0^2$ 
15     $v'_0 \leftarrow s'_0(u'_0 + u_0^2 + u_0) + u'_0 v_1 + v_0 + 1$ 
16     $[u', v'] \leftarrow [x + u'_0, v'_0] \triangleright (3S, 2M)$ 
17  else
18     $s'_1 \leftarrow 1 + u_1^2, l'_2 \leftarrow v_1^2$ 
19     $l'_1 \leftarrow u_0^2, l'_0 \leftarrow v_0^2 + d$ 
20     $w_1 \leftarrow 1/s'_1, w_0 \leftarrow l'_2 w_1 + u_1$ 
21     $u'_0 \leftarrow w_0^2, u'_1 \leftarrow w_1^2$ 
22     $w_2 \leftarrow w_1 + l'_2, w_3 \leftarrow u'_0 w_2$ 
23     $v'_1 \leftarrow (u'_1 + u'_0)(w_2 + s'_1) + w_3 + w_1 + l'_1$ 
24     $v'_0 \leftarrow w_3 + l'_0 + 1$ 
25     $[u', v'] \leftarrow [x^2 + u'_1 x + u'_0, v'_1 x + v'_0] \triangleright (1, 6S, 3M)$ 
26  end if
27 end if
```

an irreducible trinomial or pentanomial of degree 6 defined over \mathbb{F}_{2^m} . Then the field $\mathbb{F}_{2^{12m}}$ can be constructed by using an irreducible trinomial of degree 2 defined over $\mathbb{F}_{2^{6m}}$.

The curves C_d have been studied in coding theory (e.g. see [41]), and a distortion map such that $C(\mathbb{F}_{2^m}) \mapsto C(\mathbb{F}_{2^{12m}})$ is known. Normally, the random irreducible polynomials that define the fields $\mathbb{F}_{2^{6m}}$ and $\mathbb{F}_{2^{12m}}$ are chosen so that they are defined over the subfield \mathbb{F}_2 . However, it is better to carefully choose the irreducible polynomials so that applying the distortion map to a point $P \in C(\mathbb{F}_{2^m})$ simply involves manipulating the basis representation. The base field \mathbb{F}_{2^m} is constructed in the standard manner by using an irreducible polynomial of degree m over \mathbb{F}_2 . $\mathbb{F}_{2^{6m}}$ is constructed by using an irreducible pentanomial of degree 6 that is defined over \mathbb{F}_2 , given as

$$x^6 + x^5 + x^3 + x^2 + 1 = 0$$

Let $w \in \mathbb{F}_{2^6}$ be a root of this polynomial. Then a polynomial basis for the finite field $\mathbb{F}_{2^{6m}}$ is

$$\{1, w, w^2, w^3, w^4, w^5\}$$

Note that $w^7 = w^5 + w^4 + w^2 + w + 1$ and $w^8 = w + 1$. To define the quadratic extension of $\mathbb{F}_{2^{6m}}$, the irreducible trinomial of degree 2 over $\mathbb{F}_{2^{6m}}$ that is used is

$$x^2 + x + (w^5 + w^3) = 0$$

Let $s_0 \in \mathbb{F}_{2^{12}}$ be a root of this polynomial. Then a polynomial basis for the finite field $\mathbb{F}_{2^{12m}}$ is given by adjoining s_0 to the field $\mathbb{F}_{2^{6m}}$, to yield the 12-tuple

$$\{1, w, w^2, w^3, w^4, w^5, s_0, ws_0, w^2s_0, w^3s_0, w^4s_0, w^5s_0\}$$

Now define $s_1 = w^2 + w^4$ and $s_2 = w^4 + 1$. The distortion map which maps elements

of $C(\mathbb{F}_{2^m})$ to $C(\mathbb{F}_{2^{12m}})$ is given as

$$\psi(x, y) = (x + w y + s_2 x^2 + s_1 x + s_0)$$

To see why the basis for $\mathbb{F}_{2^{12m}}$ was chosen, note that applying the distortion map to a point $P = (x, y) \in C(\mathbb{F}_{2^m})$ can be computed using a single squaring in \mathbb{F}_{2^m} as

$$\psi(x, y) \begin{cases} x \mapsto \{x, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0\} \\ y \mapsto \{y + x^2, 0, x, 0, x + x^2, 0, 1, 0, 0, 0, 0\} \end{cases}$$

Note that the distortion map maps the x -coordinate of P to $\mathbb{F}_{2^{6m}}$, and the y -coordinate of P to $\mathbb{F}_{2^{12m}}$. Therefore, this distortion map supports the denominator elimination technique of Barreto et al [5]

It remains to consider the cost of arithmetic in $\mathbb{F}_{2^{6m}}$ and $\mathbb{F}_{2^{12m}}$. As addition is extremely cheap in characteristic 2, this operation is ignored in the analysis. A squaring in $\mathbb{F}_{2^{6m}}$ can be achieved extremely efficiently as

$$\{a, b, c, d, e, f\}^2 = \{a^2 + d^2 + e^2, e^2, b^2 + d^2 + f^2, d^2 + f^2, c^2, d^2\}$$

This takes only 6 squarings in \mathbb{F}_{2^m} . Multiplication is far more costly, and takes 18 multiplications in \mathbb{F}_{2^m} using the Karatsuba technique. A multiplication in $\mathbb{F}_{2^{12m}}$ is computed as

$$(a + bs_0)(c + ds_0) = (ac + bd(w^5 + w^3) + s_0((a + b)(c + d) + ac))$$

This costs 3 multiplications in $\mathbb{F}_{2^{6m}}$, as the multiplication by $(w^5 + w^3)$ can be handled by a series of additions. A squaring in $\mathbb{F}_{2^{12m}}$ is computed as

$$(a + bs_0)^2 = (a^2 + b^2(w^5 + w^3) + s_0(b^2))$$

This costs only 2 squarings in $\mathbb{F}_{2^{6m}}$. Therefore, a squaring in $\mathbb{F}_{2^{12m}}$ takes 12 squarings in \mathbb{F}_{2^m} , and a multiplication in $\mathbb{F}_{2^{12m}}$ takes 54 multiplications in \mathbb{F}_{2^m} . As multiplication in the extension fields is costly, it is imperative to try to avoid it whenever possible.

4.3.2 Octupling

Recall that Koblitz [64] gives a map for the curves C_d , such that for an element $P \in C(\mathbb{F}_{2^m})$, an explicit formula to compute a fixed scalar multiple of P is $[64]P = -\phi_{2^{12}}(P)$. Let $D \in \text{Pic}_C^0(\mathbb{F}_{2^m})$ be a divisor with a single point $P = (x_1, y_1)$ in the support, i.e. in Mumford representation $D = [x + x_1, y_1]$. Then $[64]D$ can be computed simply as $[64]D = [x + x_1^{2^{12}}, y_1^{2^{12}} + 1]$. This explicit formula provides an extremely cheap means of performing scalar multiplication on D , as it requires only 24 field squarings in \mathbb{F}_{2^m} . It would seem a good idea to use this explicit formula to compute the Tate pairing using the curves C_d , rather than use the formulae given in Algorithm 7 to repeatedly double D .

However, the curves C_d also have an octupling formula to compute $[8]D$, for any $D \in \text{Pic}_C^0(\mathbb{F}_{2^m})$. If $D = (P) - (\infty)$, this formula has the property that $[8]D = (P') - (\infty)$. P' can be computed as $P' = \sigma\phi_{2^6}(P)$, where ϕ_2 is the 2nd power Frobenius map, and σ is given as

$$\sigma(x_1, y_1) = (x_1 + 1, y_1 + x_1^2 + 1)$$

Note that applying the σ map twice in succession yields $\sigma^2 = (x_1, y_1 + 1)$. As $(x_1, y_1 + 1)$ is equal to the formula for $-P$, the opposite of P , then $\sigma^2 = -1$. Although not strictly accurate, the result of the octupling map on P is denoted as $P' = [8]P = \sigma\phi_{2^6}(P)$, and thus $[8]D = ([8]P) - (\infty)$. Therefore, for a divisor $D = [x + x_1, y_1]$, $[8]D$ can be computed as

$$[8]D = [x + (x_1 + 1)^{64}, (y_1 + x_1^2 + 1)^{64}],$$

which takes 12 squarings in \mathbb{F}_{2^m} . It is worth examining how the octupling map $[8]P$ relates

to the map $[64]P = -\phi_{2^{12}}(P)$ given by Koblitz. As $\sigma^2 = -1$, then using the octupling map twice in succession gives $[8]([8]P) = \sigma^2\phi_{2^{12}}(P) = -\phi_{2^{12}}(P)$, which is exactly the map given by Koblitz.

The octupling formula as defined applies only to a divisor with a single finite point in the support. However, it can be easily extended to general divisors. Let $D = (P_1) + (P_2) - 2(\infty)$ be a general divisor, where $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$. Using Mumford representation, D is represented as two polynomials $u(x) = x^2 + u_1x + u_0$ and $v(x) = v_1x + v_0$, such that

$$\begin{aligned} u &= x^2 + (x_1 + x_2)x + (x_1x_2) \\ v &= (y_2 + y_1)/(x_2 + x_1)x + (y_1x_2 + x_1y_2)/(x_2 + x_1) \end{aligned}$$

The goal is to compute $[8]D$ in such a way as to use the octupling formula that is given on each of the points P_1 and P_2 . By linearity

$$D' = [8]D = [8]((P_1) - (\infty)) + [8]((P_2) - (\infty)) = (P'_1) + (P'_2) - 2(\infty),$$

where

$$P'_1 = (x_1^{64} + 1, y_1^{64} + x_1^{128} + 1), P'_2 = (x_2^{64} + 1, y_2^{64} + x_2^{128} + 1)$$

The Mumford representation for $D' = [u', v']$ is given as $u'(x) = x^2 + u'_1x + u'_0$ and $v'(x) = v'_1x + v'_0$. $u'(x)$ is computed as

$$\begin{aligned} u'_1 &= x'_1 + x'_2 = (x_1 + x_2)^{64} \\ &= u_1^{64}, \\ u'_0 &= x'_1x'_2 = (x_1^{64} + 1)(x_2^{64} + 1) = (x_1x_2)^{64} + (x_1 + x_2)^{64} + 1 \\ &= (u_0 + u_1 + 1)^{64}, \end{aligned}$$

and $v'(x)$ is computed as

$$\begin{aligned}
v'_1 &= (y'_1 + y'_2)/(x'_1 + x'_2) = (y_1^{64} + x_1^{128} + y_2^{64} + x_2^{128})/(x_1^{64} + x_2^{64}) \\
&= ((y_1 + y_2)/(x_1 + x_2))^{64} + (x_1 + x_2)^{64} \\
&= (v_1 + u_1)^{64}, \\
v'_0 &= (y'_1 x'_2 + y'_2 x'_1)/(x'_1 + x'_2) \\
&= ((y_1^{64} + x_1^{128} + 1)(x_2^{64} + 1) + (y_2^{64} + x_2^{128} + 1)(x_1^{64} + 1))/(x_1^{64} + x_2^{64}) \\
&= ((y_1 x_2 + y_2 x_1)/(x_1 + x_2))^{64} + ((y_1 + y_2)/(x_1 + x_2))^{64} + (x_1 x_2)^{64} + \\
&\quad (x_1 + x_2)^{64} + 1 \\
&= (v_0 + v_1 + u_0 + u_1 + 1)^{64}
\end{aligned}$$

Algorithm 8 summarises this information, by giving complete formulae for octupling a divisor of any form in Mumford representation. This algorithm can be used for straightforward scalar multiplication of divisors on the curves C_d , as it is extremely efficient, taking at worst 24 field squarings to compute $[8]D$.

Algorithm 8 Octupling of a divisor $[u, v]$ on the curves C_d

```

INPUT  divisor  $[u, v]$ 
OUTPUT  $[u', v'] = 8[u, v]$ 
1  if  $\deg(u) = 2$  then
2     $[u', v'] \leftarrow [x^2 + u_1^{64}x + (u_1 + u_0 + 1)^{64}, (v_1 + u_1)^{64}x + (u_1 + u_0 + v_1 + v_0 + 1)^{64}]$ 
3  else if  $\deg(u) = 1$  then
4     $[u', v'] \leftarrow [x + (u_0 + 1)^{64}, (v_0 + u_0^2 + 1)^{64}]$ 
5  else
6     $[u', v'] \leftarrow [1, 0]$ 
7  end if

```

4.3.3 Using degenerate divisors

For genus 2 curves, a general (reduced) divisor $D \in \text{Pic}_C^0(\mathbb{F}_q)$ is of the form $D = (P_1) + (P_2) - 2(\infty)$, where P_1, P_2 are elements of $C(\mathbb{F}_q)$ or $C(\mathbb{F}_{q^2})$. However, certain divisors $D' \in \text{Pic}_C^0(\mathbb{F}_q)$ have only a single finite point in the support, i.e. $D' = (P) - (\infty)$. These divisors are called degenerate divisors. In general, multiplying a degenerate divisor

$D = (P) - (\infty)$ by a scalar n does not result in a divisor $[n]D = (Q) - (\infty)$, but instead in a general divisor. In fact, simply doubling $D = (P) - (\infty)$ gives the divisor $[2]D = (P) + (P) - 2(\infty)$ which is a general divisor. However, as shown in the previous section, multiplying a degenerate divisor on the curves C_d by 8 gives a degenerate divisor again.

The group arithmetic on degenerate divisors is much more efficient than for general divisors. For example, adding two general divisors takes $I, 3S, 22M$ in \mathbb{F}_{2^m} , using the formulae of Lange [70] for the genus 2 group law in characteristic 2. However, adding a degenerate divisor to a general divisor takes only $I, S, 10M$ in \mathbb{F}_{2^m} . Katagı et al [60, 61] exploit degenerate divisors in the context of scalar multiplication, by using a degenerate divisor as the “base-divisor”. This does not reduce the computational cost of the doubling operations in the double-and-add algorithm to compute the scalar multiple. However, each time an addition is performed, a general divisor is added to the initial degenerate divisor, which is cheaper than a general addition, as detailed above. Katagı et al [61] also show that solving the DLP using a degenerate divisor as the base-divisor is as intractable as using a general divisor.

Duursma and Lee [23] use degenerate divisors in the context of pairings on hyperelliptic curves. In this way, rather than use a divisor $D = (P) - (\infty)$ as one of the inputs to Miller’s algorithm, it is possible to simply use the finite point P , in a similar manner to pairing computation on elliptic curves. Pairing computation using degenerate divisors can be more efficient than using general divisors. In particular, if a degenerate divisor is used as the second argument to the Tate pairing, then it is possible to evaluate the functions in Miller’s algorithm at a single point, rather than at two points in the general case, which can be a significant saving.

In general, there is little advantage in defining the first argument to be a point as well, as a general divisor will be obtained with the first doubling in Miller’s algorithm. The benefit of having a reduced cost for addition also tends to be negligible compared to the cost of arithmetic in the extension field. However, it has been shown that an octupling operation exists on the curves C_d such that for a divisor $D = (P) - (\infty)$, then $[8]D = (P') - (\infty)$,

where $P' = \sigma\phi_{2^s}(P)$. As this octupling operation is extremely efficient, it makes sense to consider pairings on degenerate divisors. Therefore, in this chapter the pairing of degenerate divisors $D_1 = (P) - (\infty)$ and $D_2 = (Q) - (\infty)$ is examined. Computing a pairing using points, rather than divisors in Mumford representation, also allows for a simpler description which is used to optimise the pairing computation later.

Pairing based protocols that use degenerate divisors to speed up pairing computation typically require computing a pairing of general divisors as well. There are a number of different ways to compute pairings on two general divisors $D_1 = (P_1) + (P_2) - 2(\infty)$ and $D_2 = (Q_1) + (Q_2) - 2(\infty)$. First of all, a pairing can be computed on D_1 and D_2 using their Mumford representation. However, as explicit and fast algorithms will be derived later in this chapter for pairings using degenerate divisors, it is more convenient to exploit the bilinearity property of the Tate pairing by computing

$$\langle D_1, D_2 \rangle_n = \langle P_1, Q_1 \rangle_n \langle P_1, Q_2 \rangle_n \langle P_2, Q_1 \rangle_n \langle P_2, Q_2 \rangle_n$$

Therefore, computing a general divisor using this method is at worst 4 times the cost of computing a pairing using degenerate divisors. However, a number of techniques are available to improve this bound, that are largely the same as the techniques that are known to optimise the computation of multiple pairing values in the case of elliptic curves. A single accumulating variable can be shared for all the pairings, rather than have four separate variables, and thus only a single squaring over $\mathbb{F}_{2^{12m}}$ must be computed at each iteration. Any precomputation that is done need only take place once. The final exponentiation can also be shared, rather than computed after each separate pairing. However despite these optimisations, this approach is still substantially slower than the degenerate pairing.

The previous paragraphs detail how degenerate divisors can be used in pairing computation. However, there has been no discussion of when it is permissible to use degenerate divisors. Frey and Lange [29] examine these issues in detail. In particular, they state that if the group order of a supersingular curve has a sufficiently large co-factor, then it is not

possible to find a divisor $D \in \text{Pic}_C^0(\mathbb{F}_q)$ of prime order, such that D is degenerate. However, Table 4.3 gives several examples where the group order $\#\text{Pic}_C^0(\mathbb{F}_{2^m})$ for the curves C_d has a small co-factor. Frey and Lange motivate the discussion on degenerate divisors by showing how they can be used in both Identity Based Encryption and Short Signature schemes.

It is essential to test any implementation of the Tate pairing thoroughly, to ensure that it meets the required properties of a bilinear pairing. The computability property is addressed by simply implementing the algorithm efficiently, and the non-degeneracy property is met in this case by using a modified pairing. The implementation can be tested for the remaining property, that of bilinearity, by comparing the output of certain pairing computations, as will be explained later. We emphasise that this is only necessary to ensure that the implementation is correct - mathematics guarantees the bilinearity of the Tate pairing.

To generate random divisors in $\text{Pic}_C^0(\mathbb{F}_q)$, it is first necessary to generate random points on the curve C , over \mathbb{F}_{2^m} or $\mathbb{F}_{2^{2m}}$. The following solution is due to Koblitz [64]. Let $q = 2^m$ or $q = 2^{2m}$. To generate a random point $P = (x, y) \in C(\mathbb{F}_q)$, first generate a random $x \in \mathbb{F}_q$. Then the equation of the curve C_d , $y^2 + y = x^5 + x^3 + d$ has a solution $y \in \mathbb{F}_q$ if and only if the trace of the right-hand side of the equation is equal to zero, i.e.

$$\text{Tr}_{\mathbb{F}_q/\mathbb{F}_2}(x^5 + x^3 + d) = 0$$

If the trace is not equal to 0, random values for x should be repeatedly generated until this condition is met. As m is defined to be a prime (and hence odd), the y coordinate is computed using the half-trace as follows:

$$y = \sum_{j=0}^{(m-1)/2} (x^5 + x^3 + d)^{2^{2j}}$$

Once a solution y has been found, then the other root is given by $y + 1$.

As degenerate divisors are associated with points on the curve, it suffices to generate random points to construct a range of degenerate divisors for testing purposes. Two random

points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are used to construct a general divisor D as $D = (P_1) + (P_2) - 2(\infty)$. However, it is often useful to use Mumford's representation, rather than keep the finite points P_1 and P_2 in the support of D separate. This can be done by representing D as $D = [u, v]$ such that

$$\begin{aligned} u &= x^2 + (x_1 + x_2)x + (x_1x_2) \\ v &= (y_2 + y_1)/(x_2 + x_1)x + (y_1 + x_1(y_2 + y_1))/(x_2 + x_1) \end{aligned}$$

This conversion requires just 3 multiplications and 1 inversion.

To test that an implementation of the Tate pairing is bilinear, two random divisors $D_1, D_2 \in \text{Pic}_C^0(\mathbb{F}_{2^m})$ are generated. The divisor $D'_2 \in \text{Pic}_C^0(\mathbb{F}_{2^{12m}})$ is then obtained by applying the distortion map to D_2 . Let M be the final exponentiation required to compute the Tate pairing. Then the Tate pairing is computed using D_1 and D'_2 as $\langle D_1, D'_2 \rangle^M$. Scalar multiplication is then performed on D_1 using a random scalar l , to obtain the divisor $[l]D_1$. The Tate pairing is computed again as $\langle [l]D_1, D'_2 \rangle^M$. The output of the first pairing is then exponentiated to the power of l , and compared to the output of the second pairing. If the two values are equal this implies that the pairing is bilinear, as

$$\langle D_1, D'_2 \rangle^{Ml} = \langle [l]D_1, D'_2 \rangle^M$$

For a more "thorough" test of the bilinearity property, it is also possible to perform scalar multiplication on the second divisor D'_2 using a random value r , and then to compute the second pairing as $\langle [l]D_1, [r]D'_2 \rangle^M$. This can be equated with the first pairing value, raised to the power of lr .

If degenerate divisors are used, then D_1 and D_2 are associated with the finite points P_1 and P_2 , which are used as the input to Miller's algorithm. The bilinearity test involves multiplying the divisor D_1 by the scalar l . However, assuming l is a random scalar, this approach will normally yield a general divisor, rather than another degenerate divisor. In this case, the pairing can be computed using Mumford representation, or by splitting the

divisor to obtain the two finite points in the support. This can be done using the following method. Let $[l]D_1 = \{u, v\}$, where $u = x^2 + u_1x + u_0$ and $v = v_1x + v_0$. Once the x -coordinates x_1 and x_2 have been found, then y_1 and y_2 are trivially recovered by substituting x_1 and x_2 into the equation of v . Finding x_1 and x_2 requires solving the quadratic equation u (as $u = (x + x_1)(x + x_2)$). This can be done by first rewriting u in the form

$$z^2 + z = (u_0)/(u_1)^2,$$

where $z = x/u_1$. This equation can be solved by using the method given previously for generating random points on the curve. Once this is done, x_1 and x_2 can be recovered by multiplying the two roots by u_1 .

4.3.4 Octupling functions for the Tate pairing

As an efficient octupling operation has been derived for the curves C_d , it makes sense to use this operation to compute the Tate pairing. As degenerate divisors are being used, rather than general divisors, it must be shown how to derive the necessary functions that are required in Miller's algorithm from the octupling operation. Let $D_1 = (P) - (\infty)$ be the initial divisor, where $P = (x_P, y_P)$. To obtain the divisor D_8 , D_1 must be doubled continually to get D_2 , D_4 and then D_8 . For each $D_n = n(P) - n(\infty)$, an equivalent divisor D'_n is considered, such that $D_n = D'_n + (f_n)$. The function f_8 is the required function that has divisor

$$(f_8) = 8(P) - ([8]P) - 7(\infty)$$

The function f_8 is built up in stages, by extracting a function f'_n at each iteration such that $f'_n = (y + v(x))/u'(x)$, where $v(x)$ is from the Cantor composition step, and $u'(x)$ is from the Cantor reduction step.

The initial divisor $D_1 = (P) - (\infty)$ is represented in Mumford notation as

$$[u_1, v_1] = [x + x_P, y_P]$$

The function f_1 can be taken to be 1. D_1 is then doubled to obtain $D_2 = 2(P) - 2(\infty)$

This is done by using Cantor composition on $[u_1, v_1]$, which yields

$$[u_2, v_2] = [x^2 + x_P^2, (x_P^4 + x_P^2)x + y_P^2]$$

As this divisor is already reduced, then $D_2' = D_2$ and $f_2 = 1$. D_2 is then doubled to obtain

$D_4 = 4(P) - 4(\infty)$. Again, doubling $[u_2, v_2]$ using Cantor composition gives

$$[u_4, v_4] = [x^4 + x_P^4, x^3 + (x_P^8 + x_P^4)x^2 + (x_P^4)x + y_P^4]$$

This divisor is clearly not reduced, as u_4 has degree $2g$. Therefore, Cantor reduction must be applied to $[u_4, v_4]$, which yields the divisor $D_4' = [u_4', v_4']$ such that

$$[u_4', v_4'] = [x^2 + x + (x_P^{16} + x_P^8), (x_P^{16} + 1)x + (y_P^8 + x_P^8 + x_P^{24} + 1)]$$

Therefore, $D_4 = D_4' + (f_4)$, where $f_4 = (y + v_4(x))/u_4'(x)$. Finally, D_8 is obtained by doubling D_4' , using Cantor composition on $[u_4', v_4']$ to give

$$[u_8, v_8] = [u_4'(x)^2, (x_P^{32} + 1)x^2 + (x_P^{32} + x_P^{16})x + (y_P^{16} + x_P^{16} + x_P^{48} + 1)]$$

Again, this divisor is not reduced. Reducing $[u_8, v_8]$ gives a divisor $D_8' = [u_8', v_8']$ such that

$$[u_8', v_8'] = [x + (x_P^{64} + 1), y_P^{64} + x_P^{128} + 1]$$

Note that $D_8' = (P') - (\infty)$, where $P' = \sigma\phi_{2^6}(P)$, which confirms the octupling formula that was given previously. f_8' is given as $f_8' = (y + v_8(x))/u_8'(x)$, and therefore $D_8 =$

$D'_8 + (f_8)$, where

$$f_8 = f_1^{2^3} f_2^{2^2} f_4^2 f_8' = \left(\frac{y + v_4(\tau)}{u_4'(x)} \right)^2 \frac{y + v_8(\tau)}{u_8'(x)}$$

4.3.5 The final exponentiation

The group order for the curves C_d is $\#\text{Pic}_C^0(\mathbb{F}_{2^m}) = 2^{2m} \pm 2^{(3m+1)/2} + 2^m \pm 2^{(m+1)/2} + 1$. It is more convenient to use this group order, rather than a subgroup order, to compute the Tate pairing. This is because $\#\text{Pic}_C^0(\mathbb{F}_{2^m})$ has a Hamming weight of 5, which means that only 4 additions must be performed in Miller's algorithm. All cryptographic applications require a unique value, and so it is required to add the final exponentiation at some point in the protocol. Using the group order, the final exponentiation M for the curves C_d is

$$M = \frac{2^{12m} - 1}{\#\text{Pic}_C^0(\mathbb{F}_{2^m})} = \frac{2^{12m} - 1}{2^{2m} \pm 2^{(3m+1)/2} + 2^m \pm 2^{(m+1)/2} + 1}$$

Evaluating the exponentiation to M using standard exponentiation techniques in $\mathbb{F}_{2^{12m}}$ is computationally expensive. However, the cost can be greatly reduced by using the idea of Barreto et al. [5]. The exponentiation to $(2^{12m} - 1)$ can be factored as $(2^{12m} - 1) = (2^{6m} - 1)(2^{6m} + 1)$. As detailed in chapter 3, an element $x \in \mathbb{F}_{q^k}$ can be exponentiated to the power of $q^{k/2}$ using a trivial operation. In this case, an element $x = (a + bs_0) \in \mathbb{F}_{2^{12m}}$ can be exponentiated to the power of 2^{6m} as $x^{2^{6m}} = ((a + b) + bs_0)$. Therefore, the exponentiation to the power of $2^{6m} - 1$ can be computed with a single multiplication and inversion in $\mathbb{F}_{2^{12m}}$.

Once an element $x \in \mathbb{F}_{2^{12m}}$ has been exponentiated to the power of $(2^{6m} - 1)$, it has norm 1 with respect to $\mathbb{F}_{2^{6m}}$. In other words, $x^{2^{6m}+1} = 1$. This implies that $x^{-1} = x^{2^{6m}}$, and so inversion can be performed on x for free. As this property holds for any subsequent exponentiation, an expensive inversion operation need only be performed once when computing the final exponentiation. The remaining exponentiation to the power of

$(2^{6m} + 1)/\#\text{Pic}_C^0(\mathbb{F}_{2^m})$ can be factored as

$$\frac{(2^{6m} + 1)}{\#\text{Pic}_C^0(\mathbb{F}_{2^m})} = (2^m \mp 2^{(m+1)/2} + 1)(2^{3m} \mp 2^{(3m+1)/2} + 1)$$

The fact that the group order divides $2^{6m} + 1$ will be used in the next chapter to obtain an even more efficient means of computing the Tate pairing. In the meantime, the product given above can be unrolled as

$$(2^{4m} + 2^{3m} + 2^{2m+1} + 2^m + 1) \mp (2^{(m+1)/2}(2^{3m} + 2^{2m} + 2^m + 1))$$

Therefore, the entire final exponentiation to $(2^{12m} - 1)/\#\text{Pic}_C^0(\mathbb{F}_{2^m})$ can be computed with only 7 multiplications, 1 inversion, $(m + 1)/2$ squarings and some trivial Frobenius actions in $\mathbb{F}_{2^{12m}}$. As this is a relatively small computational cost, the emphasis must now be on improving the speed of Miller's algorithm itself.

4.4 Computing the Tate Pairing

In this section, it is detailed how to compute the Tate pairing efficiently using the supersingular genus 2 curves that were selected earlier, as well as all of the optimisations that were derived in the previous section.

4.4.1 Using an octic basis

Recall that the group order for the supersingular genus 2 curves $C_d: y^2 + y = x^5 + x^3 + d$ over \mathbb{F}_{2^m} , where m is odd and $d \in \{0, 1\}$, is given as

$$\#\text{Pic}_C^0(\mathbb{F}_{2^m}) = 2^{2m} \pm 2^{(3m+1)/2} + 2^m \pm 2^{(m+1)/2} + 1$$

Therefore, using the group order in Miller's algorithm to compute the Tate pairing implies a loop size of $2m$ iterations, assuming a standard double-and-add algorithm. This algorithm can be combined with the simplified explicit formulae for doubling elements of

$\text{Pic}_C^0(\mathbb{F}_{2^m})$ as given in Algorithm 7. The resulting algorithm yields an efficient pairing implementation on both degenerate and general divisors. However, in the previous section it was shown that these curves support an extremely efficient octupling operation. Explicit formulae were then derived for the intermediate functions that are required in Miller's algorithm in the case of degenerate divisors. Given a divisor $D = (P) - (\infty)$, where $P = (x_P, y_P)$, the function associated with $[8]D$ is

$$f_{8P}(x, y) = \left(\frac{y + v_4(x)}{u_4'(x)} \right)^2 \frac{y + v_8(x)}{u_8'(x)},$$

where

$$\begin{aligned} v_4(x) &= x^3 + (x_P^8 + x_P^4)x^2 + (x_P^4)x + y_P^4, \\ v_8(x) &= (x_P^{32} + 1)x^2 + (x_P^{32} + x_P^{16})x + (y_P^{16} + x_P^{16} + x_P^{48} + 1) \end{aligned}$$

Note that the denominator $u_4'(x)^2 u_8'(x)$ is evaluated only at x , which is the x -coordinate of the distorted image point. The distortion map ψ defined previously maps the x -coordinate of the point to the field $\mathbb{F}_{2^{6m}}$. Therefore, the denominator is also defined over $\mathbb{F}_{2^{6m}}$ and is eliminated by the final exponentiation.

The goal is to compute the Tate pairing $\langle P, \psi(Q) \rangle$ on the degenerate divisors $D_1 = (P) - (\infty)$ and $D_2 = (Q) - (\infty)$. At each iteration of the algorithm, the point P is octupled and the function given above is calculated. As a result, one would expect Miller's algorithm to have $2m/3$ iterations, as opposed to $2m$ iterations when doubling. However, there are a number of problems with this approach when a pairing is computed on degenerate divisors. The group order $\#\text{Pic}_C^0(\mathbb{F}_{2^m})$ has a Hamming weight of 5, meaning that four additions must take place in Miller's algorithm. However, the final addition yields a vertical line function and does not need to be computed as a result [23].

The first problem is that the number of doublings that must be performed between each addition in the group order might not be a multiple of 3. If this is the case, then either one or two extra doublings must be performed. However, even a single doubling of a

degenerate divisor $D = (P) - (\infty)$ results in the general divisor $[2]D = 2(P) - 2(\infty)$. The second problem relates to the additions that must be performed. In the overwhelmingly common case, any addition performed on a degenerate divisor will yield a general divisor. Therefore, the computational advantage of using a degenerate divisor as the first input to Miller's algorithm is quickly negated.

Here we propose an alternative approach which overcomes these problems. Let $D = (P) - (\infty)$ be the first input divisor to the Tate pairing. It is required to construct a function f , such that $(f) = [N]D = [N]((P) - (\infty))$, where N is defined to be the group order $\#P_{1C}^0(\mathbb{F}_{2^m})$ in this case. Rather than construct f in stages as in Miller's algorithm, note that f is composed of several intermediate functions. Let D_{2^i} be the reduced divisor equivalent to $[2^i]((P) - (\infty))$, and let f_{2^i} be the function such that

$$[2^i]((P) - (\infty)) = D_{2^i} + (f_{2^i})$$

Let D' be a reduced divisor equivalent to $D_{2^{2m}} \pm D_{2^{(3m+1)/2}}$, and let h_1 be the function that arises from this addition process such that

$$D_{2^{2m}} \pm D_{2^{(3m+1)/2}} = D' + (h_1)$$

Similarly, let h_2 be the function that arises from the addition of D' with D_{2^m} , and let h_3 be the function from the addition (subtraction) of the reduced divisor equivalent to $D' + D_{2^m}$ with $D_{2^{(m+1)/2}}$. The final function that arises from the addition with D can be omitted as it is eliminated by the final exponentiation. The function f is then constructed as

$$f = f_{2^{2m}} f_{2^{(3m+1)/2}} f_{2^m} f_{2^{(m+1)/2}} h_1 h_2 h_3$$

Therefore, rather than compute f using a double-and-add algorithm, it is possible to compute each f_{2^i} that is required separately. No additions take place when computing any of these values, which removes the need for conditional statements inside the loop. The

additions that are required to compute the h_i functions are performed afterwards, and thus the problems associated with using degenerate divisors are avoided. If the f_{2^i} functions are calculated separately, the number of iterations of Miller's algorithm will far exceed $2m/3$, thus defeating the purpose of this optimisation. A better strategy is to have a single octupling loop up to $2m/3$. Then whenever the index reaches the required value, the function at that point can be saved. These function values can then be multiplied together after the loop.

As an octupling is performed at each loop iteration, the function values must be saved at the nearest index to i . However, depending on the value of i , one or two extra doublings might have to take place to get the correct value for f_{2^i} . Let $D' = (P') - (\infty)$ be the degenerate divisor that corresponds to the function that is saved in the loop. It has been shown previously that doubling a degenerate divisor does not contribute any line function to the accumulating function. Therefore, if an extra doubling must be performed, it is only required to square the function to obtain the correct value for f_{2^i} .

A further doubling yields the divisor $[4]D' = 4(P') - 4(\infty)$. The explicit line function for this divisor was given previously as

$$y + v_4(x) = y + x^3 + (x_{P'}^8 + x_{P'}^4)x^2 + (x_{P'}^4)x + y_{P'}^4$$

Therefore, when two additional doublings must be performed, it suffices to square the function twice, and then to multiply it by the function given above. Rather than evaluate this function at $\psi(Q)$, where $Q = (x_Q, y_Q)$, it is possible to build the distortion map ψ into the function. This is given here in the basis defined earlier for $\mathbb{F}_{2^{12m}}$. The constant term is

$$\{y_Q + x_Q^2(1 + x_Q + x_{P'}^8 + x_{P'}^4) + x_{P'}^4 x_Q + y_{P'}^4\},$$

and the remaining terms are

$$\{x_Q^2 + x_{P'}^4, x_{P'}^8 + x_{P'}^4, 1, x_Q^2 + x_Q, 0, 1, 0, 0, 0, 0\}$$

Finally, the functions h_1 , h_2 and h_3 that correspond to the additions in the group order must be calculated. This is easily done by adding the relevant divisors D_{2^i} and extracting the functions that are required from the addition process. It is a relatively simple matter to construct the divisors D_{2^i} . Firstly, the convention that $x^{2^i} = x^{(i)}$ is adopted for convenience. Given a point $P = (x_P, y_P)$, then $[2^{3^i}]P$ can be derived by generalising the octupling formula. The x -coordinate of $[2^{3^i}]P$ is given as

$$x_P^{(6^i)} + \gamma_1(i),$$

where $\gamma_1(i)$ is 1 when i is odd and 0 otherwise. The y -coordinate of $[2^{3^i}]P$ is given as

$$y_P^{(6^i)} + \gamma_1(i)x_P^{(6^{i+1})} + \gamma_3(i),$$

where $\gamma_3(i) = 1$ when $i \equiv 1, 2 \pmod{4}$, and 0 otherwise. The exponents in brackets in these formulae are reduced modulo m , as $x^{2^m} = x$ for $x \in \mathbb{F}_{2^m}^*$. However, the i in D_{2^i} may not be an exact multiple of 3. This is easily solved by performing at most an extra two doublings using the approach described previously. The divisors D_{2^i} are then added together using Cantor composition and reduction, and the relevant functions are extracted and multiplied to obtain f .

4.4.2 Precomputing the first point

At each iteration of the octupling algorithm to compute the f_{2^i} functions, it is necessary to octuple the iterating point P , and to evaluate the explicit functions that were derived previously at the image point $\psi(Q)$. It has been shown that the denominator of this function does not need to be computed, as it is eliminated by the final exponentiation. Therefore, the function that must be computed at each iteration is the product $\alpha\beta$, where $\alpha = (y + v_4(x))^2$ and $\beta = (y + v_8(x))$. These functions are evaluated at $\psi(Q)$, which is the point that results from the application of the distortion map ψ to the point $Q \in C(\mathbb{F}_{2^m})$. Let $P = (x_P, y_P)$

Then $v_4(x)$ and $v_8(x)$ are given as

$$\begin{aligned} v_4(x) &= x^3 + (x_P^8 + x_P^4)x^2 + (x_P^4)x + y_P^4, \\ v_8(x) &= (x_P^{32} + 1)x^2 + (x_P^{32} + x_P^{16})x + (y_P^{16} + x_P^{16} + x_P^{48} + 1) \end{aligned}$$

It is important to optimise the generation and evaluation of the $\alpha\beta$ product, as it is computed at each iteration of the loop. The first optimisation is to build the distortion map into the formulae for α and β . This means that the second input point to the Tate pairing is now $Q = (x_Q, y_Q) \in C(\mathbb{F}_{2^m})$, rather than $\psi(Q) \in C(\mathbb{F}_{2^{12m}})$. Recall that the distortion map is $\psi(x, y) = (x + w, y + s_2x^2 + s_1x + s_0)$, where $s_0^2 + s_0 = w^5 + w^3$, $s_1 = w^2 + w^4$ and $s_2 = w^4 + 1$. Then

$$\begin{aligned} (y + v_4(x)) &= y_Q + s_2x_Q^2 + s_1x_Q + s_0 + (x_Q + w)^3 + (x_P^8 + x_P^4)(x_Q + w)^2 + \\ &\quad (x_P^4)(x_Q + w) + y_P^4 \end{aligned}$$

Squaring this function gives

$$\begin{aligned} \alpha &= y_Q^2 + s_2^2x_Q^4 + s_1^2x_Q^2 + s_0^2 + x_Q^6 + x_Q^4w^2 + x_Q^2w^4 + w^6 + \\ &\quad (x_P^{16} + x_P^8)(x_Q^4 + w^4) + (x_P^8)(x_Q^2 + w^2) + y_P^8 \end{aligned}$$

The basis for $\mathbb{F}_{2^{12m}}$ was constructed to avoid the need for explicitly calculating elements such as w^4 . Therefore, the formula for α must be rewritten in terms of this basis. Let $s_2^2 = w$, $s_1^2 = w^4 + w + 1$ and $s_0^2 = s_0 + w^5 + w^3$. Then the constant component of α , written in the basis for $\mathbb{F}_{2^{12m}}$, is

$$\{y_Q^2 + x_Q^2 + x_Q^6 + 1 + (x_P^{16} + x_P^8)x_Q^4 + x_P^8x_Q^2 + y_P^8\},$$

and the remaining components are

$$\{x_Q^4 + x_Q^2, x_Q^4 + 1 + x_P^8, 0, x_P^{16} + x_P^8, 0, 1, 0, 0, 0, 0, 0\}$$

Similarly, applying the distortion map to $\beta = (y + v_8(r))$ gives

$$\begin{aligned} \beta = & y_Q + s_2 x_Q^2 + s_1 x_Q + s_0 + (x_P^{32} + 1)(x_Q + w)^2 + (x_P^{32} + x_P^{16})(x_Q + w) + \\ & (y_P^{16} + x_P^{16} + x_P^{48} + 1) \end{aligned}$$

Expressing β in the basis derived for the field $\mathbb{F}_{2^{12m}}$ has constant component

$$\{y_Q + (x_P^{32})x_Q^2 + (x_P^{32} + x_P^{16})x_Q + y_P^{16} + x_P^{16}(1 + x_P^{32}) + 1\}$$

and the remaining components are

$$\{x_P^{32} + x_P^{16}, x_Q + x_P^{32} + 1, 0, x_Q^2 + x_Q, 0, 1, 0, 0, 0, 0, 0\}$$

It is possible to precompute any power of x_Q that is required for both α and β , as these terms are constant throughout the algorithm. However, as the point P is octupled at each iteration, it is required to constantly update the values of x_P and y_P in α and β . Note that the values required for x_P and y_P in α and β are generally of the form $x_P^{2^i}$ for some i . However, there are only m possible values for i , as $x_P^{2^m} = x_P$. Therefore, rather than explicitly octuple P , it is better to precompute all of the possible powers of x_P and y_P . These powers can be stored in an array and then accessed in the algorithm using array indexing. Firstly, two arrays of size m are instantiated. Each index i in the arrays then consists of the value $x_P^{2^i}$ and $y_P^{2^i}$. This precomputation requires $2m$ squarings in \mathbb{F}_{2^m} , which is a relatively small cost.

The formulae given for α and β must then be rewritten, so that the required powers for x_P and y_P can be accessed from the precomputed arrays, rather than from the explicit

octupling of P . This can be done by using the formulae given for computing $[2^{3i}]P$. Firstly, α is examined. Recall that $\gamma_1(i)$ is 1 when i is odd and 0 otherwise, and $\gamma_3(i) = 1$ when $i \equiv 1, 2 \pmod{4}$, and 0 otherwise. The constant term of α is then

$$\{y_Q^2 + x_Q^6 + (x_P^{(6i+4)} + x_P^{(6i+3)})x_Q^4 + (x_P^{(6i+3)} + 1 + \gamma_1(i))x_Q^2 + y_P^{(6i+3)} + \gamma_1(i)x_P^{(6i+4)} + \gamma_3(i) + 1\}$$

and the remaining components are

$$\{x_Q^4 + x_Q^2, x_Q^4 + x_P^{(6i+3)} + \gamma_1(i) + 1, 0, x_P^{(6i+4)} + x_P^{(6i+3)}, 0, 1, 0, 0, 0, 0, 0\},$$

where $x^{(i)} = x^{2^i}$ as before. Similarly, for β the constant term is

$$\{y_Q + x_P^{(6i+5)}(x_Q + x_Q^2) + x_P^{(6i+4)}(x_P^{(6i+5)} + x_Q + \gamma_1(i) + 1) + y_P^{(6i+4)} + \gamma_3(i) + \gamma_1(i)x_Q^2 + 1\},$$

and the remaining components are

$$\{x_P^{(6i+5)} + x_P^{(6i+4)}, x_Q + x_P^{(6i+5)} + \gamma_1(i) + 1, 0, x_Q^2 + x_Q, 0, 1, 0, 0, 0, 0, 0\}$$

Therefore, α and β are constructed at each iteration of the loop with only two multiplications in \mathbb{F}_{2^m} each, assuming that all of the powers of x_Q and y_Q that are required are precomputed. This is a large saving on having to explicitly calculate the required powers of x_P and y_P at each iteration. The remaining task is to show how to multiply α and β in an efficient manner. A general multiplication in $\mathbb{F}_{2^{12m}}$ is an expensive operation as it costs 54 multiplications in \mathbb{F}_{2^m} . However, both α and β are of the form $(a + bw + cw^2 + dw^4 + s_0)$. It is possible to derive a special multiplication routine that exploits the sparse structure of both α and β . This routine is derived in Appendix A.3, and costs only 11 multiplications in \mathbb{F}_{2^m} using some Karatsuba-like optimisations.

4.4.3 Absorbing powers of 8

As described in the previous subsection, the functions α and β are calculated at each iteration of the octupling algorithm. These functions are first multiplied together using the explicit routine given in Appendix A.3, and then with the accumulating variable f . However, it is necessary to first exponentiate the accumulating variable $f \in \mathbb{F}_{2^{12m}}$ to the power of 8. This can be achieved with three squarings in $\mathbb{F}_{2^{12m}}$. As a squaring in $\mathbb{F}_{2^{12m}}$ takes 12 squarings in \mathbb{F}_{2^m} , the total cost per iteration of the algorithm is 36 squarings in \mathbb{F}_{2^m} .

However, it is possible to avoid these squarings by building the exponentiation into the α and β terms inside the algorithm. This technique is feasible for finite fields of low characteristic, and was introduced to pairing based cryptography by Duursma and Lee [23]. The first function that must be computed is $f_{2^{(m+1)/2}}$. This can be obtained by $(m-1)/6$ iterations of the octupling algorithm and a doubling. Therefore, for the index $i = 0$ to $(m-1)/6$, α and β must be raised to the power of $2^{(m-7-6i)/2}$, to avoid octupling the accumulating variable f .

The next function to be calculated is f_{2^m} . This can be obtained with $(m-1)/3$ iterations of the octupling algorithm and a further doubling. So, for the index $i = (m-1)/6$ to $(m-1)/3$, α and β must be exponentiated to $2^{(m-4-3i)}$. However, further work is necessary if the functions that were previously calculated are to be reused. The exponentiation used to calculate $f_{2^{(m+1)/2}}$ is $2^{(m-7-6i)/2}$. Therefore, to reuse this function it is necessary to exponentiate it to the power of $2^{(m-1)/2}$, as $(2^{(m-7-6i)/2})^{(2^{(m-1)/2})} = 2^{(m-4-3i)}$.

The function $f_{2^{(3m+1)/2}}$ is computed with $(m-1)/2$ iterations of the octupling algorithm and two doublings. For the index $i = (m-1)/3$ to $(m-1)/2$, this involves exponentiating α and β to the power of $2^{(3m-9-6i)/2}$. As before, some extra work must be done to reuse the functions that have been previously calculated. Finally, the function $f_{2^{2m}}$ is computed with $(2m-2)/3$ iterations of the octupling algorithm and two doublings. For the index $i = (m-1)/2$ to $(2m-2)/3$, α and β must be exponentiated to the power of $2^{(2m-5-3i)}$, again with some further work to reuse the functions.

Therefore, the algorithm to compute each of the four f_{2^i} functions consists of four

separate loops of $(m - 1)/6$ iterations each. Each loop contains separate formulae, each of which is obtained by exponentiating the formulae for α and β given previously to the required power. This requires the precomputation of $x_Q^{2^i}$ and $y_Q^{2^i}$, for every $0 < i < m$, and using array indexing in the formulae. It costs two multiplications in \mathbb{F}_{2^m} to construct each term, which is exactly the same computational cost as that given previously for α and β . The actual process of exponentiating α and β will be examined in more detail in the following chapter.

Each time a function is reused, it is necessary to perform $(m - 1)/2$ squarings, some multiplications as well as some applications of the Frobenius endomorphism in $\mathbb{F}_{2^{12m}}$. Performing these operations four times negates any of the advantages associated with eliminating the octupling of the accumulating variable. However, it is better to group all of the operations that require the exponentiation to $2^{(m-1)/2}$ together, so that this powering is only computed once. As any application of the Frobenius map can be trivially computed, the additional cost of this optimisation is only $(m - 1)/2$ squarings and 7 multiplications in $\mathbb{F}_{2^{12m}}$.

The total cost of this optimisation is as follows. $2m$ additional squarings in \mathbb{F}_{2^m} must be performed in the precomputation stage. Then, $6m - 6$ squarings in \mathbb{F}_{2^m} and 378 multiplications in \mathbb{F}_{2^m} are computed after the loops. The previous strategy of squaring the accumulating variable three times per iteration costs roughly $24m$ squarings in \mathbb{F}_{2^m} in total, as the loop size is approximately $2m/3$. Therefore, this optimisation saves roughly $18m$ squarings, at the cost of 378 multiplications. For values of m that are used in practice, this optimisation is faster than that given in the previous subsection. However, the complexity of implementing this approach is considerable.

4.5 Experimental Results

In this section, experimental data is provided to validate our assertion that efficient pairing calculation is possible on genus 2 curves over \mathbb{F}_{2^m} . The first task in implementing any of

the techniques given in this chapter is to select m . There are a number of conditions on the selection of m that must be satisfied so that the implementation of the Tate pairing is invulnerable to attack, as discussed at the start of this chapter. Firstly, m must be chosen so that $\#\text{Pic}_C^0(\mathbb{F}_{2^m})$ has a large prime factor. Secondly, $\text{Pic}_C^0(\mathbb{F}_{2^m})$ must be large enough to resist any generic attack against the DLP in this group. Thirdly, as the embedding degree of the genus 2 curves is $k = 12$, a further condition on m is that $\mathbb{F}_{2^{12m}}^*$ is large enough to resist any sub-exponential time attack on the DLP.

Various examples of m that satisfy all of these conditions for the two curves C_0 $y^2 + y = x^5 + x^3$ and C_1 $y^2 + y = x^5 + x^3 + 1$ were given earlier in Table 4.3. The values that were chosen for m range from $m = 79$ to $m = 239$. All of these values are large enough to satisfy the security considerations detailed above. Therefore, the smallest values of m are chosen for each curve, as the larger m is, the more computationally expensive the arithmetic in both $\text{Pic}_C^0(\mathbb{F}_{2^m})$ and $\mathbb{F}_{2^{12m}}^*$. For the curve C_0 the value for m is $m = 103$, and for the curve C_1 the value is $m = 79$. As will be detailed later, these parameters have the advantage that an element of \mathbb{F}_{2^m} can be represented inside a single hardware register (assuming our computing platform).

Table 4.4 details the experimental results on the supersingular genus 2 curve C_1 over $\mathbb{F}_{2^{79}}$. Table 4.5 details the experimental results for the curve C_0 over $\mathbb{F}_{2^{103}}$. The first three cases in each table give timings for the implementation of the Tate pairing using the group order for each curve. All three cases in each table share the fast means of performing finite field arithmetic as detailed previously, as well as the efficient method to compute the final exponentiation. The first case in both tables is an algorithm to compute the Tate pairing where both input elements are general divisors. A standard right-to-left doubling algorithm is used, where the iterating divisor is doubled using the explicit formulae given previously in Algorithm 7.

The second case in both tables uses degenerate divisors and octupling to compute the Tate pairing. This algorithm precomputes the relevant powers of the first input point to avoid explicitly octupling at each iteration. The third case in both tables also uses degenerate

Table 4 4 Experimental results for the curve C_1 over $\mathbb{F}_{2^{79}}$

Case	Description	Running time (ms)
1	General divisors	11 17
2	Degenerate I	2 16
3	Degenerate II	1 89
4	Octupling	0 41

Table 4 5 Experimental results for the curve C_0 over $\mathbb{F}_{2^{103}}$

Case	Description	Running time (ms)
1	General divisors	11 90
2	Degenerate I	3 11
3	Degenerate II	2 69
4	Octupling	0 638

divisors and octupling to compute the Tate pairing. However, it precomputes powers of the second input point to avoid having to octuple the accumulating variable each iteration. Finally, the fourth case in both tables gives the timing for scalar multiplication using the octupling formulae given in Algorithm 8. The scalars in question are m bits long and have a random Hamming weight. This timing is included to compare the cost of computing the Tate pairing with that of scalar multiplication on the curves C_d . All of the timings are given in milliseconds.

The general conclusion that can be drawn from these tables is that the Tate pairing can be computed efficiently on (supersingular) genus 2 curves over \mathbb{F}_{2^m} . A striking conclusion from the tables is that the degenerate divisor case yields a far more efficient implementation than the general case. This result shows that it is worth using cryptographic protocols in the genus 2 setting that take advantage of pairing computation on degenerate divisors, as detailed by Frey and Lange [29]. It can be seen that the optimisation of avoiding the octupling of the accumulating variable by building the exponentiation into the intermediate functions inside the algorithm gives a slight improvement over the simpler method. Finally, it can be concluded that pairing calculation on genus 2 curves over \mathbb{F}_{2^m} is not as efficient as scalar multiplication, although the difference between them is narrower than was previously

thought

All of the experiments were performed on our platform of a Pentium IV, which has a clock speed of 2.8 GHz, and which runs version 2.6.15 of the Linux kernel. The code is written in C/C++ and is compiled using version 4.03 of the GCC/G++ compiler suite. The efficient implementation of the finite field \mathbb{F}_{2^m} is taken from MIRACL 5.01. Recall that m was chosen so that an element of \mathbb{F}_{2^m} can be represented inside a single hardware register, rather than using a multi-precision representation. More precisely, arithmetic in \mathbb{F}_{2^m} is performed in the 128-bit registers available to the Pentium IV, which support the SSE2 instruction set. SSE2 is a SIMD (Single Instruction, Multiple Data) instruction set, meaning that a single SSE2 instruction manipulates the entire 128-bit register. This offers improved performance over the use of four separate 32-bit registers. In particular, this instruction set enables the multiplication of elements in \mathbb{F}_{2^m} that is about twice as fast as a standard multi-precision implementation.

4.6 Conclusion

In this chapter, it was shown that pairing calculation on supersingular genus 2 curves over \mathbb{F}_{2^m} is efficient, and is a valid candidate for the practical implementation of pairing based protocols as a result. Firstly, it was shown how to select curves with the maximum embedding degree permitted for genus 2 curves. It was shown how to select m such that the relevant security parameters are satisfied, and formulae were given to double elements in $\text{Pic}_C^0(\mathbb{F}_{2^m})$ that exploit the special form of the curve equations that were chosen.

It was detailed how to construct the extension fields that are required. The use of degenerate divisors in pairing computation was then explored. An octupling automorphism was derived, and explicit formulae were given for the intermediate functions that are required in Miller's algorithm. It was shown how to compute the final exponentiation efficiently. Then the actual implementation of the Tate pairing on these curves was detailed. It was shown how to avoid the problems associated with using degenerate divisors, and how to use

precomputation to speed up the algorithm

Finally, experimental results were given. It was demonstrated that it is possible to compute the Tate pairing using general divisors efficiently. If degenerate divisors are used, then the implementation is particularly efficient. However, the results given in this chapter should be viewed more as a proof of concept, rather than in a practical manner. This is because the methods used to compute the genus 2 Tate pairing in this chapter are superseded by simpler and faster methods that will be described in the following chapter.

Chapter 5

The η_T Pairing

5.1 Introduction

In the previous chapter, an efficient implementation of the Tate pairing was described on the supersingular genus 2 curves $C_d: y^2 + y = x^5 + x^3 + d$ over \mathbb{F}_{2^m} , where $d \in \{0, 1\}$. These curves have the maximum embedding degree for supersingular genus 2 curves of $k = 12$. As the group order for these curves is approximately $\#\text{Pic}_C^0(\mathbb{F}_{2^m}) \approx 2^{2m}$, Miller's algorithm requires around $2m/3$ iterations if the fast octupling operation on C_d is exploited. However, implementing this algorithm efficiently is difficult due to the additions that must be performed.

In this chapter, it is shown how it is possible to achieve an implementation of the Tate pairing on the same genus 2 curves by using the η pairing construct. The η pairing requires m iterations of Miller's algorithm using the octupling operation in the genus 2 case, as opposed to $2m/3$ iterations using the straightforward method. However, the arithmetic associated with the η pairing is substantially less expensive to compute, and thus it yields an efficient pairing implementation. The resulting algorithm is approximately as efficient as the method given in the previous chapter, but is much simpler to describe and to implement.

It is then shown how for certain supersingular elliptic curves it is possible to avoid the final exponentiation when computing the η pairing, as long as the vertical line functions are

included in the pairing computation. This is the first time that a method has been given to compute the Tate pairing for cryptographic purposes that has no final exponentiation. This technique is not useful in practice, as the evaluation of the vertical line functions in Miller's algorithm is costly. However, it is likely that future research will find an application for this idea.

It is then shown how it is possible to achieve a more efficient pairing computation on the genus 2 curves C_d by using the η_T pairing. This pairing has a more complicated final exponentiation than the η pairing, and an extra addition must be performed after the loop. However, it has approximately half the number of loop iterations of the η pairing. Various techniques are detailed to achieve the fastest implementation possible. Finally, a comprehensive series of tests is conducted on the various pairing implementations, and the chapter is concluded.

This chapter contains joint work with Paulo S L M Barreto, Steven D Galbraith and Michael Scott, which has been accepted for publication in *Designs, Codes and Cryptography*. A preprint is available at the ePrint archive as Barreto et al [4]. This chapter also contains joint work with Steven D Galbraith and Caroline Sheedy, which has been accepted for publication in the *Journal of Mathematical Cryptology*. A preprint of this paper is available at the ePrint archive as Galbraith et al [33].

5.2 The Theory of the η_T Pairing

Recall that Duursma and Lee [23] introduce several optimisations which lower the computational cost of the Tate pairing. These techniques apply to supersingular hyperelliptic curves of the form $C: y^2 = x^p - x + d$ over \mathbb{F}_{p^m} , where $d = \pm 1$, $p \equiv 3 \pmod{4}$ and m and $2p$ are co-prime. These curves have an embedding degree of $k = 2p$. Rather than using the group order $\#\text{Pic}_C^0(\mathbb{F}_{p^m})$ to compute the Tate pairing, Duursma and Lee propose using a multiple of the group order of the form $N = p^{pm} + 1$. This order has Hamming weight 2 in base p , meaning that only a single addition must be performed in Miller's al-

gorithm. However, Duursma and Lee show that as the addition takes place on the final iteration of the algorithm, the line function that corresponds to the addition is a vertical line function. Therefore, the addition does not need to be computed, leading to the elimination of conditional logic from the algorithm.

As the order used to compute the Tate pairing is $N = p^{pm} + 1$, the final exponentiation is computed as

$$M = (q^k - 1)/N = (p^{2pm} - 1)/(p^{pm} + 1) = p^{pm} - 1$$

which can be trivially computed with a multiplication and inversion in $\mathbb{F}_{p^{2pm}}$. Duursma and Lee also introduce the idea of pairing computation on degenerate divisors and hence points on the curve, as covered in the previous chapter. Instead of deriving the intermediate line functions in Miller's algorithm from the Cantor composition and reduction of the iterating divisor, Duursma and Lee provide explicit formulae which depend only on the coordinates of the original input point. It is also shown how to absorb the exponentiation to p in Miller's algorithm into the explicit formulae.

Therefore, the proposed pairing implementation has a trivial final exponentiation and the arithmetic inside the loop can be computed efficiently. However, these optimisations come at the cost of a longer loop of pm iterations. Duursma and Lee's most significant contribution is to show how the number of iterations of the loop can be shortened from pm to m . This is done by absorbing the exponentiation to p inside the explicit formulae. The resulting algorithm enables an extremely efficient implementation of the Tate pairing.

Unfortunately, there are very few specific hyperelliptic curves of the form given by Duursma and Lee that have a suitable embedding degree over a finite field of low characteristic. In fact, only the elliptic curves $y^2 = x^3 - x + d$ over \mathbb{F}_{3^m} with an embedding degree of $k = 6$ are interesting for pairing based cryptography. Kwon [66] transfers the ideas of Duursma and Lee to the elliptic curves $y^2 + y = x^3 + x + d$ over \mathbb{F}_{2^m} , where $d \in \{0, 1\}$ and the embedding degree is $k = 4$. In joint work with Barreto, Galbraith and Scott [4],

we show how to generalise and extend the Duursma and Lee loop shortening approach to arbitrary supersingular curves by introducing the η_T pairing construct

Let C be a supersingular (hyperelliptic) curve over \mathbb{F}_q , where $q = p^m$, with an embedding degree of $k > 1$. Also, let ψ be a distortion map on the curve C which enables the denominator elimination technique. Let N be the order that is used to compute the Tate pairing, which can either be the group order, a multiple of the group order, or a prime subgroup order. Let $D_1, D_2 \in \text{Pic}_C^0(\mathbb{F}_q)$ have order dividing N , and $[N]D_1 = (f_N)$, for some function f . Then the level N Tate pairing is computed as

$$\langle D_1, \psi(D_2) \rangle_N = f_N(\psi(D_2))$$

The η_T pairing is not a new bilinear pairing but simply an alternative means of computing the Tate pairing on certain supersingular curves. The main idea of the η_T pairing is to compute the Tate pairing using an order $T \in \mathbb{Z}$, such that

$$\eta_T(D_1, \psi(D_2)) = f_T(\psi(D_2))$$

Unlike the straightforward method to compute the Tate pairing, the condition that $[T]D_1 = (\infty)$ can be dropped. In order to get the loop reduction idea of Duursma and Lee, the goal is to select a value for T that is smaller than N . However, the vast majority of choices for T will not yield a non-degenerate, bilinear pairing. The following lemma gives a method for selecting T such that the η_T pairing fulfils all of the properties of a bilinear pairing. However, this lemma merely shows what values of T yield a bilinear pairing, it does not show how to select T such that $T < N$. The key requirement is an automorphism on the curve, such as the octupling automorphism derived in the previous chapter.

Lemma 4 *Let $\langle D_1, \psi(D_2) \rangle_N$ be the Tate pairing as defined above. Then if L, a and T are co-prime to N the η_1 pairing is a non-degenerate bilinear pairing such that*

$$(\eta_1(D_1, D_2))^M)^{aT^{a-1}} = (\langle D_1, \psi(D_2) \rangle_N^M)^L,$$

where the following properties hold

- 1 $[T]D \equiv \gamma(D)$ where γ is an automorphism of C which is defined over \mathbb{F}_q
- 2 γ and ψ satisfy the condition $\gamma\psi^q = \psi$
- 3 $T^a + 1 = LN$ for some $a \in \mathbb{N}$ and $L \in \mathbb{Z}$
- 4 $T = q + cN$ for some $c \in \mathbb{Z}$

For a proof of this lemma, see Barreto et al [4]

5.3 The Genus 2 η Pairing

The aim of this section is to select a value for T such that the η_T pairing is bilinear and non-degenerate for the supersingular genus 2 curves that were considered in the previous chapter. If T can be chosen to be approximately equivalent to 2^{2m} , then it may be possible to derive a pairing implementation that is as efficient as the octupling algorithm given in the previous chapter.

5.3.1 Finding a suitable value for T

Consider the curves $C_d: y^2 + y = x^5 + x^3 + d$ over \mathbb{F}_{2^m} , where $d \in \{0, 1\}$. As was detailed in the previous chapter, these curves have the maximum embedding degree for a supersingular genus 2 curve of $k = 12$. Recall that for an element $D \in \text{Pic}_C^0(\mathbb{F}_{2^m})$, such that $D = (P) - (\infty)$, it is possible to octuple the divisor D as $[8]D = (P') - (\infty)$, where $P' = \sigma\phi_{2^6}(P)$, ϕ_2 is the 2nd power Frobenius map and the map σ is given as

$$\sigma(x, y) = (x + 1, y + x^2 + 1),$$

where $\sigma^2 = -1$. A distortion map for C_d is

$$\psi(x, y) = (x + w, y + s_2x^2 + s_1x + s_0),$$

which supports the denominator elimination technique, as the x -coordinate is mapped to the quadratic subfield of $\mathbb{F}_{2^{12m}}$

The first condition to be met in selecting T is to find an automorphism γ of C_d which is defined over \mathbb{F}_q , such that $[T]D \equiv \gamma(D)$. The obvious candidate is the map $[8]P = [2^3]P = \sigma\phi_{2^6}(P)$. However, using the value $T = 2^3$ does not yield a bilinear pairing. Instead, consider the map $[2^{3m}]P = \sigma^m\phi_{2^{6m}}(P) = \sigma^m(P)$. Then $\gamma = \sigma^m$ is also an automorphism on the curves C_d defined over \mathbb{F}_q . Normally, $q = 2^m$ denotes the base-field. However, here the notation $q = 2^{3m}$ is used to take advantage of the fast octupling operation in computing the η_T pairing. Therefore, T is defined as $T = q = 2^{3m}$.

The next step is to prove that the automorphism γ and the distortion map ψ satisfy the condition that $\gamma\psi^q = \psi$. Let $q = 2^{3m}$ as above, where m is prime and $\gamma = \sigma^m$. Recall that $w^8 = w + 1$, $s_1 = w^2 + w^4$, $s_2 = w^4 + 1$ and $s_0^2 + s_0 = w^5 + w^3$. Then $s_1^8 = s_1$, $s_2^8 = s_2 + 1$ and $s_0^8 = s_0 + w^2$. As m is prime, it must be congruent to either 1 or 3 modulo 4. Firstly, let $m \equiv 1 \pmod{4}$ and thus $3m \equiv 3 \pmod{12}$. Then

$$\begin{aligned} \gamma\psi^q(x, y) &= \sigma^m\psi^{2^{3m}}(x, y) \\ &= \sigma\psi^{2^3}(x, y) \\ &= \sigma(x + w + 1, y + (s_2 + 1)x^2 + s_1x + s_0 + w^2) \\ &= (x + w, y + s_2x^2 + s_1x + s_0) \\ &= \psi(x, y) \end{aligned}$$

Let $w^{2^9} = w + 1$, $s_1^{2^9} = s_1$, $s_2^{2^9} = s_2 + 1$ and $s_0^{2^9} = s_0 + w^2 + 1$. Now let $m \equiv 3 \pmod{4}$

and thus $3m \equiv 9 \pmod{12}$. Then

$$\begin{aligned}
\gamma\psi^q(x, y) &= \sigma^m \psi^{2^{3m}}(x, y) \\
&= \sigma^3 \psi^{2^9}(x, y) \\
&= -\sigma(x + w + 1, y + (s_2 + 1)x^2 + s_1x + s_0 + w^2 + 1) \\
&= (x + w, y + s_2x^2 + s_1x + s_0) \\
&= \psi(x, y)
\end{aligned}$$

Therefore, an automorphism γ has been derived for the curves C_d such that $\gamma\psi^q = \psi$ and $[T]D \equiv \gamma(D)$, where $T = q = 2^{3m}$. It remains to satisfy the conditions that $T^a + 1 = LN$ for $a \in \mathbb{N}$ and $L \in \mathbb{Z}$, and $T = q + cN$ for $c \in \mathbb{Z}$. As $T = q$, it follows that $c = 0$. In the previous chapter, the fact that the group order for the curves C_d divides $2^{6m} + 1$ was used to derive an efficient means of computing the final exponentiation. Let $N = 2^{6m} + 1$, in which case the equation $T^a + 1 = LN$ is satisfied as $(2^{3m})^2 + 1 = (1)(2^{6m} + 1)$, and hence $a = 2$ and $L = 1$. The final exponentiation is $M = (2^{12m} - 1)/(2^{6m} + 1) = 2^{6m} - 1$, which can be computed with a conjugation with respect to $\mathbb{F}_{2^{6m}}$, a multiplication and an inversion in $\mathbb{F}_{2^{12m}}$.

Thus all the conditions of the η_T pairing have been met, and the resulting bilinear pairing is computed as

$$(\eta_T(D_1, D_2))^M = (\langle D_1, \psi(D_2) \rangle_N)^M$$

Computing the η_T pairing in this way with $T = q = 2^{3m}$ requires $3m$ iterations of Miller's algorithm. However, as an octupling operation is used to compute the intermediate functions, only m iterations are required. Computing the Tate pairing with $N = 2^{6m} + 1$ and using the octupling operation requires $2m$ iterations of Miller's algorithm. As detailed in the previous chapter, the Tate pairing can be computed using the group order and octupling in $2m/3$ iterations. However, this approach does not have the benefits of the η_T pairing,

such as a cheap final exponentiation and no conditional logic inside the loop

5.3.2 Optimising the arithmetic

The η_T pairing when $T = q = 2^{3m}$ can be viewed as a generalisation of Duursma and Lee's idea to the genus 2 curves C_d . To distinguish this case from an even faster instantiation of the η_T pairing later in this chapter, the subscript T is dropped when $T = q$. Hence, this pairing is referred to as the η pairing. Given a divisor $D = (P) - (\infty)$, where $P = (x_P, y_P)$, the function $f_{8,P}$ associated with $[8]D$ was derived in the previous chapter. As the denominator of this function can be discarded due to the form of the distortion map, $f_{8,P}$ is given as

$$f_{8,P}(x, y) = (y + v_4(x))^2(y + v_8(x)),$$

where

$$\begin{aligned} v_4(x) &= x^3 + (x_P^8 + x_P^4)x^2 + (x_P^4)x + y_P^4 \\ v_8(x) &= (x_P^{32} + 1)x^2 + (x_P^{32} + x_P^{16})x + (y_P^{16} + x_P^{16} + x_P^{48} + 1) \end{aligned}$$

The notation $\alpha\beta$ is used for this intermediate function evaluated at the distorted image point, where $\alpha = (y + v_4(x))^2$ and $\beta = (y + v_8(x))$

An explicit formula to compute the η pairing on the genus 2 curves C_d using the input divisors $D_1 = (P) - (\infty)$ and $D_2 = (Q) - (\infty)$ is then given as

$$\eta(P, Q) = \prod_{i=0}^{m-1} f_{8,[8^i]P}(\psi(Q))^{8^{m-1-i}}$$

As detailed in the previous chapter, a number of optimisations are available to expedite the computation of $\alpha\beta$ at each iteration of the loop. Rather than compute the various powers of x_P and y_P that are required at each iteration, it is possible to precompute all of the possible values and to store them in an array. This approach costs $2m$ squarings before the loop, but

the cost of calculating the functions inside the loop reduces to only 4 multiplications per iteration of the loop

In order to avoid octupling the accumulating variable at each iteration of the algorithm, the exponentiation can be absorbed into the formulae inside the loop. To do this, α must be replaced with $\alpha^{2^{3(m-1-i)}}$ and β with $\beta^{2^{3(m-1-i)}}$, which involves precomputing powers of ι_Q and y_Q . In Appendix A 1, it is shown how to derive explicit formulae for these terms. This optimisation replaces $3m$ squarings in $\mathbb{F}_{2^{12m}}$ with $2m$ squarings in \mathbb{F}_{2^m} , which is a large improvement. Finally, it is possible again to exploit the sparseness of the α and β terms, by unrolling the multiplication using Karatsuba-like techniques, as detailed in Appendix A 3. The resulting algorithm to compute the η pairing is given in Algorithm 9.

We now examine the computational cost of the algorithm in terms of operations in \mathbb{F}_{2^m} . The precomputation takes $4m$ squarings in \mathbb{F}_{2^m} . At each iteration of the algorithm, the calculation of the α and β functions takes only 4 multiplications in \mathbb{F}_{2^m} . The multiplication of α and β using the Karatsuba approach takes 11 multiplications in \mathbb{F}_{2^m} . Finally, this value is multiplied with the accumulating variable f , which takes an expensive 54 multiplications in \mathbb{F}_{2^m} . Therefore, for a loop size m , the total computational cost is $4m$ squarings and $69m$ multiplications in \mathbb{F}_{2^m} . The final exponentiation is trivially computed as a multiplication, an inversion, a squaring and a few Frobenius actions in $\mathbb{F}_{2^{12m}}$. Experimental results for the genus 2 η pairing are given later in this chapter.

5.4 Avoiding the Final Exponentiation

It is also shown in our paper [4] how to compute the η pairing using supersingular elliptic curves over characteristic 2 and 3. In this section, we show how the final exponentiation required to compute the η pairing for the characteristic 2 elliptic case can be omitted if the vertical line functions are included in Miller's algorithm. This is the first time that it has been demonstrated that the Tate pairing can be computed without a final exponentiation for a cryptographically useful exponent. This observation also holds true for the genus 2 case,

Algorithm 9 The genus 2 η pairing

INPUT $D_1 = (P) - (\infty), D_2 = (Q) - (\infty) \in \text{Pic}_C^0(\mathbb{F}_{2^m}), P = (x_P, y_P), Q = (x_Q, y_Q)$ OUTPUT $f = \eta(P, Q) \in \mathbb{F}_{2^{12m}}$

```
1  ▷ Initialisation set  $\gamma = 1$  if  $m \equiv 1 \pmod{4}$ , otherwise  $\gamma = 0$ 
2  ▷ Precompute powers of P and Q
3   $x_1[i] \leftarrow x_P^{2^i}, y_1[i] \leftarrow y_P^{2^i}, x_2[i] \leftarrow x_Q^{2^i}, y_2[i] \leftarrow y_Q^{2^i}, 0 < i < m - 1$ 
4   $f \leftarrow 1$ 
5
6  for  $i = 0$  to  $m - 1$  do
7    ▷ All  $k_*$  in the next 2 lines to be considered modulo  $m$ 
8     $k_1 \leftarrow (3m - 3 - 3i), k_2 \leftarrow (k_1 + 1), k_3 \leftarrow (k_2 + 1)$ 
9     $k_4 \leftarrow (3i), k_5 \leftarrow (k_4 + 1), k_6 \leftarrow (k_5 + 1)$ 
10
11   ▷ Calculate  $\alpha \leftarrow a + bw + cw^2 + dw^4 + s_0$ 
12    $d \leftarrow x_1[k_4] + x_1[k_5]$ 
13    $a \leftarrow y_2[k_2] + (x_1[k_4] + 1 + x_2[k_3]) \quad x_2[k_2] + d \quad x_2[k_3] + y_1[k_4] + \gamma$ 
14    $b \leftarrow x_2[k_3] + x_2[k_2]$ 
15    $c \leftarrow r_2[k_3] + r_1[k_4] + 1$ 
16
17   ▷ Calculate  $\beta \leftarrow e + f_2w + gw^2 + hw^4 + s_0$ 
18    $f_2 \leftarrow x_1[k_5] + x_1[k_6]$ 
19    $e \leftarrow y_2[k_1] + f_2 \quad x_2[k_1] + y_1[k_5] + x_1[k_6] \quad (x_1[k_5] + x_2[k_2]) + x_1[k_5] + \gamma$ 
20    $g \leftarrow x_2[k_1] + x_1[k_6] + 1$ 
21    $h \leftarrow x_2[k_2] + x_2[k_1]$ 
22
23   ▷ Unroll  $\alpha\beta$  multiplication using Karatsuba
24    $dh \leftarrow d \quad h, dg \leftarrow d \quad g, ch \leftarrow c \quad h, cg \leftarrow c \quad g, ae \leftarrow a \quad e, bf_2 \leftarrow b \quad f_2$ 
25    $t_0 \leftarrow ae + ch + dg + dh$ 
26    $t_1 \leftarrow (a + b)(f_2 + e) + ae + bf_2 + dh$ 
27    $t_2 \leftarrow (a + c)(g + e) + ae + cg + bf_2 + ch + dg$ 
28    $t_3 \leftarrow (b + c)(g + f_2) + bf_2 + cg + ch + dg + 1$ 
29    $t_4 \leftarrow (a + d)(h + e) + ae + dh + cg$ 
30    $t_5 \leftarrow (b + d)(h + f_2) + bf_2 + dh + ch + dg + 1$ 
31    $u_1 \leftarrow (t_0, t_1, t_2, t_3, t_4, t_5, a + e + 1, b + f_2, c + g, 0, d + h, 0)$ 
32    $f \leftarrow f \cdot u_1$ 
33 end for
34
35 ▷ Perform the final exponentiation
36  $f \leftarrow f^{(2^{6m}-1)(2)(2^{3m})}$ 
```

however we have been unable to prove this as yet. Instead, we include the proof for the supersingular elliptic case in characteristic 2.

These curves are given by an equation of the form $E_d: y^2 + y = x^3 + \iota x + d$ over \mathbb{F}_{2^m} , where $d \in \{0, 1\}$, and have an embedding degree of $k = 4$. Kwon [66] shows how to transfer the Duursma-Lee techniques to compute the Tate pairing efficiently on these curves. In our paper [4] it is (independently) shown how to compute the η pairing efficiently using these curves, which results in essentially the same algorithm as that given by Kwon. A distortion map is $\psi(x, y) = (x + s^2, y + sx + t)$, where $s \in \mathbb{F}_{2^2}$ satisfies $s^2 + s + 1 = 0$, and $t \in \mathbb{F}_{2^4}$ satisfies $t^2 + t + s = 0$. Note that the x -coordinate is mapped to a subfield of $\mathbb{F}_{2^{4m}}$, and hence the denominator elimination technique applies. These curves support a simple doubling formula, such that for a point $P = (x_P, y_P) \in E(\mathbb{F}_{2^m})$, then

$$[2^i]P = (x_P^{2^{2i}} + \iota, y_P^{2^{2i}} + \iota x_P^{2^{2i}} + \tau(\iota)),$$

where $\tau(\iota) = 1$ if $\iota \equiv 2, 3 \pmod{4}$ and zero otherwise. The group order for the curves E_d is given as $\#E_d(\mathbb{F}_{2^m}) = 2^m \pm 2^{(m+1)/2} + 1$. Rather than use this order to compute the Tate pairing, the η pairing can be computed on these curves using a loop of m iterations with no additions. After this loop, an exponentiation to the power of $2^{2m} - 1$ must be performed to obtain a unique value suitable for cryptographic use. This can be easily computed using a conjugation with respect to $\mathbb{F}_{2^{2m}}$, a multiplication and an inversion in $\mathbb{F}_{2^{4m}}$. After the exponentiation has been applied, the resulting element of $\mathbb{F}_{2^{4m}}$ has order dividing $2^{2m} + 1$. As $2^{2m} + 1$ is the norm map with respect to $\mathbb{F}_{2^{2m}}$, the pairing value is said to be an element of norm 1.

There is no need to include the vertical line functions when the distortion map ψ is used, as they are eliminated by the final exponentiation to $2^{2m} - 1$. However, here we show that the final exponentiation can be omitted, as long as the vertical line functions are included in Miller's algorithm. Firstly, it is shown how to construct the field $\mathbb{F}_{2^{4m}}$. The extension field $\mathbb{F}_{2^{2m}}$ is defined by the irreducible polynomial $s^2 + s + 1 = 0$, where $s \in \mathbb{F}_{2^2}$. Similarly,

the field $\mathbb{F}_{2^{4m}}$ is defined as a quadratic extension of $\mathbb{F}_{2^{2m}}$ using the irreducible polynomial $t^2 + t + s = 0$, where $t \in \mathbb{F}_{2^4}$. The conjugate of an element $\alpha = (a + bt) \in \mathbb{F}_{2^{4m}}$ (with respect to $\mathbb{F}_{2^{2m}}$) is written as $\bar{\alpha} = (a + b) + bt$. Also, the norm of α (with respect to $\mathbb{F}_{2^{2m}}$) is written as

$$(a + bt)\overline{(a + bt)} = a^2 + ab + b^2s$$

The conjugate of a function was defined previously in Chapter 3. Let $h(Q)$ be a function that is evaluated at a point $Q \in E(\mathbb{F}_{q^k})$. Then the conjugate of $h(Q)$, denoted $\overline{h(Q)}$ is equal to $h(-Q)$, where $-Q$ is the opposite of Q . Let $l_P(\tau, y) = y - y_P - \lambda(\tau - \tau_P)$ be the equation of the tangent to the curve at P , and let $v_P(x) = x - x_P$ be the vertical line through P . Then it is well known that

$$l_P(\psi(Q))\overline{l_P(\psi(Q))} = v_P(\psi(Q))^2 v_{[2]P}(\psi(Q))$$

The η pairing for the elliptic curves E_d is written as the product

$$z = \prod_{i=0}^{m-1} \left(\frac{l_{[2^i]P}(\psi(Q))}{v_{[2^{i+1}]P}(\psi(Q))} \right)^{2^{m-1-i}}$$

The aim is to show that z has norm 1 (with respect to $\mathbb{F}_{2^{2m}}$), and hence $z\bar{z} = 1$, where $\bar{z} = z^{2^{2m}}$. Using the equality given above, $z\bar{z}$ is written as

$$z\bar{z} = \prod_{i=0}^{m-1} \left(\frac{l_{[2^i]P}(\psi(Q))\overline{l_{[2^i]P}(\psi(Q))}}{v_{[2^{i+1}]P}(\psi(Q))^2} \right)^{2^{m-1-i}} = \prod_{i=0}^{m-1} \left(\frac{v_{[2^i]P}(\psi(Q))^2 v_{[2^{i+1}]P}(\psi(Q))}{v_{[2^{i+1}]P}(\psi(Q))^2} \right)^{2^{m-1-i}}$$

This can be simplified by cancelling the $v_{[2^{i+1}]P}(\psi(Q))$ terms as

$$z\bar{z} = \prod_{i=0}^{m-1} \frac{v_{[2^i]P}(\psi(Q))^{2^{m-1-i}}}{v_{[2^{i+1}]P}(\psi(Q))^{2^{m-1-i}}}$$

By setting $j = i + 1$, this can be written as

$$z\bar{z} = \left(\prod_{i=0}^{m-1} v_{[2^i]P}(\psi(Q))^{2^{m-i}} \right) / \left(\prod_{j=1}^m v_{[2^j]P}(\psi(Q))^{2^{m-j}} \right) = \frac{v_P(\psi(Q))^{2^m}}{v_{[2^m]P}(\psi(Q))} = 1,$$

since

$$v_P(\psi(Q))^{2^m} = (x_Q + s^2 + x_P)^{2^m} = x_Q + s^2 + x_P + 1 = v_{[2^m]P}(\psi(Q))$$

Therefore, as the η pairing has norm 1 with respect to the quadratic subfield, assuming the vertical line functions are included, the final exponentiation can be omitted

It remains to determine how the value of the η pairing without any final exponentiation is related to the value of the η pairing when the final exponentiation is included. Let $z \in \mathbb{F}_{2^{4m}}$ be the output of the η pairing when vertical lines are included in Miller's algorithm. As it has been shown that z is an element of norm 1 with respect to $\mathbb{F}_{2^{2m}}$, then $z^{2^{2m}+1} = 1$. This implies that $z^{2^{2m}} = z^{-1}$, and $z^{2^{2m}-1} = z^{-2}$. Therefore

$$\left(\prod_{i=0}^{m-1} \left(\frac{l_{[2^i]P}(\psi(Q))}{v_{[2^{i+1}]P}(\psi(Q))} \right)^{2^{m-1-i}} \right)^{2^{2m}-1} = \left(\prod_{i=0}^{m-1} \left(\frac{l_{[2^i]P}(\psi(Q))}{v_{[2^{i+1}]P}(\psi(Q))} \right)^{2^{m-1-i}} \right)^{-2}$$

However, as exponentiating to the power of $2^{2m} - 1$ removes the need to compute the vertical line functions, this can be rewritten as

$$\left(\prod_{i=0}^{m-1} (l_{[2^i]P}(\psi(Q)))^{2^{m-1-i}} \right)^{2^{2m}-1} = \left(\prod_{i=0}^{m-1} \left(\frac{l_{[2^i]P}(\psi(Q))}{v_{[2^{i+1}]P}(\psi(Q))} \right)^{2^{m-1-i}} \right)^{-2}$$

This equivalence shows how the two different methods to compute the η pairing are related

Therefore, it has been shown how to avoid computing the final exponentiation to $2^{2m} - 1$, at the cost of losing the optimisation of denominator elimination. The exponentiation to $2^{2m} - 1$ can be computed essentially as a multiplication and an inversion in $\mathbb{F}_{2^{4m}}$. By repeatedly taking the norm with respect to the quadratic subfields of $\mathbb{F}_{2^{4m}}$, it is possible to

reduce the inversion in $\mathbb{F}_{2^{4m}}$ efficiently to an inversion in \mathbb{F}_{2^m} . In contrast, the algorithm that includes the vertical line functions requires the computation of an inversion at each iteration of Miller’s algorithm. This can be avoided by using the two-variable approach of Galbraith et al. [31]. However, this technique involves extra arithmetic at each iteration of the loop, as well as an inversion after the loop. Therefore, it can be concluded that there is no benefit to using the new pairing algorithm introduced in this section, when compared to the denominator elimination technique.

However, if it can be shown that it is possible to avoid the final exponentiation in circumstances where it is expensive to compute, then this new technique may find a practical use. In our paper [33], it is shown by a similar proof that the final exponentiation can also be avoided when computing the η pairing using the (Duurmsa-Lee) supersingular hyperelliptic curves $y^2 = x^p - x + d$ over \mathbb{F}_{p^m} , where $d = \pm 1$ and $p \equiv 3 \pmod{4}$. This technique also applies to the genus 2 η pairing.

5.5 The Genus 2 η_T Pairing

It has been shown that the genus 2 η pairing yields an efficient and simple pairing implementation, which is approximately as efficient as the implementation of the Tate pairing given in the previous chapter. The genus 2 η pairing uses the value $T = q = 2^{3m}$ to compute the pairing, which results in a loop of m iterations using the fast octupling operation on the curves C_d . However, it remains to be seen whether the theory of the η_T pairing permits the selection of a value for T that is smaller than $q = 2^{3m}$. In this section, this open question is addressed in the affirmative. The convention to drop the subscript T when $T = q$ was previously described. In this section, the pairing implementation is simply referred to as the η_T pairing, as the subscript T denotes “truncated”.

5.5.1 Finding a suitable value for T

Recall that four conditions must be satisfied to implement the η_I pairing. The first condition is that $[T]D \equiv \gamma(D)$, where γ is an automorphism of the curve which is defined over \mathbb{F}_q . For the η pairing on the supersingular genus 2 curves C_d , this condition is satisfied by setting $T = q = 2^{3m}$ and $\gamma = \sigma^m$, where $\sigma(x, y) = (x + 1, y + x^2 + 1)$. It was shown that the second condition of the η_I pairing, that $\gamma\psi^q = \psi$, holds for the η pairing on the genus 2 curves C_d , where ψ is a distortion map on the curve that supports the denominator elimination technique. It was then shown that instead of using a multiple of the group order $N = 2^{6m} + 1$, the value $T = q = 2^{3m}$ can be used to compute the η pairing.

The goal of this section is to find a value for T such that T is smaller than $q = 2^{3m}$, and that the η_T pairing using this value is bilinear and non-degenerate. The first condition that must be satisfied is that $[T]D \equiv \gamma(D)$. Recall that the group order for the curves C_d is given as

$$\#\text{Pic}_C^0(\mathbb{F}_{2^m}) = 2^{2m} \pm 2^{(3m+1)/2} + 2^m \pm 2^{(m+1)/2} + 1$$

Rather than use the multiple $N = (2^{6m} + 1)$ of $\#\text{Pic}_C^0(\mathbb{F}_{2^m})$, as for the η pairing, consider the multiple N such that

$$N = (2^m \mp 2^{(m+1)/2} + 1)(\#\text{Pic}_C^0(\mathbb{F}_{2^m})) = 2^{3m} \pm 2^{(3m+1)/2} + 1$$

Let $q = 2^{3m}$ as before. Note that

$$q - N = 2^{3m} - (2^{3m} \pm 2^{(3m+1)/2} + 1) = \mp 2^{(3m+1)/2} - 1$$

Define $T = q - N = \mp 2^{(3m+1)/2} - 1$. Then

$$[T]D = [q - N]D = [q]D - [N]D = [q]D = \gamma(D)$$

Therefore, the condition that $[T]D \equiv \gamma(D)$ is satisfied by using the automorphism $\gamma = \sigma^m$. As this is the same value for γ that was used to compute the η pairing, the second condition that $\gamma\psi^q = \psi$ is already proven. The condition that $T = q + \ell N$, where $\ell \in \mathbb{Z}$, is easily seen to satisfy $\ell = -1$, as $T = q - N$. The remaining condition that must be satisfied on T is $T^a + 1 = LN$, for some $a \in \mathbb{N}$ and $L \in \mathbb{Z}$. Let $a = 2$. Then

$$T^2 + 1 = 2(2^{3m} \pm 2^{(3m+1)/2} + 1) = 2N$$

Therefore, $a = L = 2$. Thus, all the conditions of the η_T pairing have been satisfied, and a bilinear pairing on $D_1, D_2 \in \text{Pic}_C^0(\mathbb{F}_{2^m})$ is computed as

$$(\eta_T(D_1, D_2)^M)^T = \langle D_1, \psi(D_2) \rangle_N^M,$$

where the final exponentiation is $M = (2^{12m} - 1)/(2^{3m} \pm 2^{(3m+1)/2} + 1)$

Rather than compute the η_T pairing using the divisor D_1 and the order $T = -2^{(3m+1)/2} - 1$, it is convenient to use the opposite of D_1 and the order $-T = 2^{(3m+1)/2} + 1$ to avoid performing an inversion. Therefore, T is now defined as $T = 2^{(3m+1)/2} \pm 1$. Computing the η_T pairing in this manner implies a loop size of $(3m + 1)/2$ iterations. However, observe that T can be written as $T = 2^{3(m-1)/2+2} \pm 1$. Therefore, the η_T pairing can be computed as $(m - 1)/2$ octuplings, as well as two doublings and an addition. This is clearly far superior to the m loop iterations of the η pairing, as well as the $2m/3$ loop iterations of the method given in the previous chapter.

5.5.2 Optimising the arithmetic

There are two obstacles to implementing the genus 2 η_T pairing efficiently, namely the complicated final exponentiation and the final doublings and addition. Firstly, the final exponentiation is examined. In the η case, the final exponentiation can be computed essentially as a multiplication and an inversion in $\mathbb{F}_{2^{12m}}$. In the η_T case, the final exponentiation

is to the power of MT . Recall that M is given as

$$M = \frac{2^{12m} - 1}{N} = \frac{(2^{6m} - 1)(2^{6m} + 1)}{2^{3m} \pm 2^{(3m+1)/2} + 1}$$

This expression can be simplified by using the fact that

$$2^{6m} + 1 = (2^{3m} \pm 2^{(3m+1)/2} + 1)(2^{3m} \mp 2^{(3m+1)/2} + 1)$$

Therefore, the exponentiation to M is computed as

$$M = (2^{6m} - 1)(2^{3m} \mp 2^{(3m+1)/2} + 1)$$

The entire final exponentiation is then written as

$$\begin{aligned} MT &= (2^{6m} - 1)(2^{3m} \mp 2^{(3m+1)/2} + 1)(\mp 2^{(3m+1)/2} - 1) \\ &= (2^{6m} - 1)(2^{3m} \mp 2^{4m} 2^{(m+1)/2} - 1) \end{aligned}$$

The exponentiation to $(2^{6m} - 1)$ can be achieved with a conjugation with respect to $\mathbb{F}_{2^{6m}}$, as well as a multiplication and an inversion in $\mathbb{F}_{2^{12m}}$. The remaining exponentiation can be computed in $(m + 1)/2$ squarings, 2 multiplications and a few Frobenius actions in $\mathbb{F}_{2^{12m}}$. Therefore, it has been shown how to compute the final exponentiation required for the η_T method in a relatively inexpensive way.

The second disadvantage of the genus 2 η_T pairing, when compared to the η pairing, is that two doublings and an addition must be performed after the loop of $(m - 1)/2$ iterations. However, various techniques can be used to reduce the penalty of performing these operations. Let $D_1 = (P) - (\infty)$ be the degenerate divisor that is used as the input to the η_T pairing. The final addition that must be performed is an addition of divisors $[2^{(3m+1)/2}]D_1$ and $[\pm 1]D_1$, where $[\pm 1]D_1$ is a degenerate divisor of the form $[\pm 1]D_1 = ([\pm 1]P) - (\infty)$. However, it will be shown that the final addition does not need to be performed in the case

that D_1 is a degenerate divisor, as it does not contribute to the pairing value

The order of the divisor $D_1 = (P) - (\infty)$ divides $N = 2^{3m} \pm 2^{(3m+1)/2} + 1$, as N is a multiple of the group order. Using the formula derived for the genus 2 η pairing, $[2^{3m}]D_1$ is a degenerate divisor of the form $[2^{3m}]D_1 = \sigma^m(P) - (\infty)$. Therefore the final divisor is given as

$$[2^{(3m+1)/2} \pm 1]D_1 \equiv [N]D_1 - [2^{3m}]D_1 \equiv (\mp \sigma^m(P)) - (\infty)$$

Therefore, the divisor $[2^{(3m+1)/2}]D_1$ is given as

$$\begin{aligned} [2^{(3m+1)/2}]D_1 &= [2^{(3m+1)/2} \pm 1]D_1 - [\pm 1]D_1 \\ &= (\mp \sigma^m(P)) - (\infty) - ([\pm 1]P) - (\infty) \\ &= (\mp \sigma^m(P)) + (\mp P) - 2(\infty) \end{aligned}$$

Therefore, when the final addition is performed in the degenerate case, the composition operation in Cantor's algorithm cancels out the $(\mp P)$ by using a vertical line function, which can be omitted from the algorithm.

The final two doublings are now considered, again in the case for a degenerate divisor $D_1 = (P) - (\infty)$. The divisor after the octupling loop is $D' = [2^{3(m-1)/2}]D_1 = (P') - (\infty)$. Doubling this divisor using the Cantor composition algorithm yields a divisor of the form $[2]D' = 2(P') - 2(\infty)$. As this divisor is already reduced it does not contribute any line function to the accumulating function. It is known from the explicit formulae derived in the previous chapter that the line function that comes from the doubling of a special divisor of the form $2(P') - 2(\infty)$ is

$$l(\tau, y) = y + \tau^3 + (x_{P'}^8 + \tau_{P'}^4)x^2 + (x_{P'}^4)\tau + y_{P'}^4$$

Therefore, to compute the operations that are required after the octupling loop, it suffices to square the accumulating variable f twice, and then to multiply it by the function above

In the previous chapter it was shown how to build the distortion map into this function, in order to avoid manipulating points over the extension field $\mathbb{F}_{2^{12m}}$.

Therefore, it has been shown how to compute both the final exponentiation and the final addition/doublings efficiently. An explicit formula to compute the η_T pairing on the curves C_d using the input divisors $D_1 = (P) - (\infty)$ and $D_2 = (Q) - (\infty)$ is

$$\eta_T(P, Q) = \left(\prod_{i=0}^{(m-3)/2} f_{8^i}[\mathfrak{s}]_P(\psi(Q))^{2^{(3m-9-6i)/2}} \right)^{2^2} l(\psi(Q)),$$

where l is the function defined above. All of the optimisations used to compute the η pairing for the curves C_d can be used to compute the η_T pairing. To avoid octupling the accumulating variable, α and β must be replaced with $\alpha^{2^{3(m-3-2i)/2}}$ and $\beta^{2^{3(m-3-2i)/2}}$. It is shown how to derive these formulae in Appendix A.2.

Therefore, it has been shown how the octupling loop, the final operations and the final exponentiation can be computed efficiently. The algorithm to compute the genus 2 η_T pairing is given in Algorithm 10 in the case that $m = 103$. For other values of m it suffices only to change the formula for extracting the current point after the loop. The cost of the precomputation is $4m$ squarings in \mathbb{F}_{2^m} , which is the same as for the η pairing. The cost of the arithmetic at each iteration of the loop is 69 multiplications in \mathbb{F}_{2^m} , again the same as for the η pairing. However, the loop is only of length $(m-1)/2$ instead of m for the η pairing. Note that computing the final doubling function takes only two multiplications in \mathbb{F}_{2^m} , as all of the required powers of (x_P, y_P) and (x_Q, y_Q) are precomputed. Experimental results for the genus 2 η_T pairing are given in the following section.

5.6 Experimental Results

In this section, experimental results are provided to verify our claim that it is possible to compute the Tate pairing efficiently using the genus 2 η and η_T pairings. The first task in implementing either pairing is to select suitable values for m . Recall that the prime m must

Algorithm 10 The genus 2 η_I pairing when $m = 103$

INPUT $D_1 = (P) - (\infty), D_2 = (Q) - (\infty) \in \text{Pic}_C^0(\mathbb{F}_{2^m}), P = (x_P, y_P), Q = (x_Q, y_Q)$ OUTPUT $f = \eta_I(P, Q) \in \mathbb{F}_{2^{12m}}$

```
1  ▷ Initialisation set  $\gamma = 1$  if  $m \equiv 1 \pmod{4}$ , otherwise  $\gamma = 0$ 
2  ▷ Precompute powers of P and Q
3   $x_1[i] \leftarrow x_P^{2^i}, y_1[i] \leftarrow y_P^{2^i}, x_2[i] \leftarrow x_Q^{2^i}, y_2[i] \leftarrow y_Q^{2^i}, 0 \leq i \leq m-1$ 
4   $f \leftarrow 1$ 
5
6  for  $i = 0$  to  $(m-3)/2$  do
7    ▷ All  $k_*$  in the next 2 lines to be considered modulo  $m$ 
8     $k_1 \leftarrow (3m-9-6i)/2, k_2 \leftarrow (k_1+1), k_3 \leftarrow (k_2+1)$ 
9     $k_4 \leftarrow (3m-3+6i)/2, k_5 \leftarrow (k_4+1), k_6 \leftarrow (k_5+1)$ 
10
11   ▷ Calculate  $\alpha \leftarrow a + bw + cw^2 + dw^4 + s_0$ 
12    $d \leftarrow x_1[k_4] + x_1[k_5]$ 
13    $a \leftarrow y_2[k_2] + (x_1[k_4] + 1 + x_2[k_3]) x_2[k_2] + d x_2[k_3] + y_1[k_4] + \gamma$ 
14    $b \leftarrow x_2[k_3] + x_2[k_2]$ 
15    $c \leftarrow r_2[k_3] + r_1[k_4] + 1$ 
16
17   ▷ Calculate  $\beta \leftarrow e + f_2w + gw^2 + hw^4 + s_0$ 
18    $f_2 \leftarrow x_1[k_5] + x_1[k_6]$ 
19    $e \leftarrow y_2[k_1] + f_2 x_2[k_1] + y_1[k_5] + x_1[k_6] (x_1[k_5] + x_2[k_2]) + x_1[k_5] + \gamma$ 
20    $g \leftarrow x_2[k_1] + x_1[k_6] + 1$ 
21    $h \leftarrow x_2[k_2] + x_2[k_1]$ 
22
23    $f \leftarrow f (\alpha \beta)$ 
24 end for
25
26 ▷ "Extract" current point  $(x_P, y_P)$ 
27  $x_P \leftarrow x_1[m-3] + 1$ 
28  $y_P \leftarrow y_1[m-3] + x_1[m-2]$ 
29
30 ▷ Perform the final doublings/addition
31  $t \leftarrow (y_2[0] + x_2[1] (1 + x_2[0] + x_1[3] + x_1[2]) + x_1[2] x_2[0] + y_1[2])$ 
32  $f \leftarrow f^4 (t, r_2[1] + r_1[2], r_1[3] + r_1[2] - 1, x_2[1] + x_2[0], 0, 1, 0, 0, 0, 0)$ 
33
34 ▷ Perform the final exponentiation
35  $f \leftarrow f^{(2^{6m}-1)(2^{3m}-2^{4m}2^{(m+1)/2}-1)}$ 
```

be chosen so that $\text{Pic}_C^0(\mathbb{F}_{2^m})$ has a large prime factor, and that $\text{Pic}_C^0(\mathbb{F}_{2^m})$ and $\mathbb{F}_{2^{km}}^*$ are large enough to resist any attack on the DLP in these groups. In the previous chapter, the value $m = 79$ was chosen for the curve $C_1: y^2 + y = x^5 + x^3 + 1$, and $m = 103$ was chosen for the curve $C_0: y^2 + y = x^5 + x^3$. These parameters fulfil the security conditions detailed above, and have the advantage that an element of \mathbb{F}_{2^m} can be represented by a single machine word, which greatly aids in the efficient implementation of the underlying field arithmetic.

The second task is to select other pairing implementations against which to compare the efficiency of the algorithms given in this chapter. In the previous chapter, it was shown how to implement the Tate pairing efficiently using the group order of the curves C_d . As the same values for m are used in both chapters, it is possible to compare the experimental results directly. However, in order to gauge the true efficiency of pairing implementation using supersingular genus 2 curves over low characteristic, it is necessary to give experimental results for an equivalent implementation on an elliptic curve. To give an accurate comparison with the genus 2 case, the elliptic curve in question should be supersingular and defined over a finite field of characteristic 2.

The natural candidates are the elliptic curves $E_d: y^2 + y = x^3 + x + d$ over \mathbb{F}_{2^m} , where $d \in \{0, 1\}$, that were used previously in this chapter. In our paper [4], it is shown how to implement the η_I pairing efficiently on these curves. To compare against the genus 2 implementations, suitable prime values for m must be chosen so that km is roughly the same for both cases. In this way, the output of the η_T pairing for both curves has the same resistance against attacks on the DLP (as it is an element in $\mathbb{F}_{2^{km}}^*$). Two security levels are defined, following from the values for km in the genus 2 case. Firstly, for the value $m = 79$ in the genus 2 case, a suitable value for m in the elliptic case is $m = 239$, as then $km \approx 950$ for both curves. Secondly, to compare with the value $m = 103$ in the genus 2 case, $m = 313$ is chosen in the elliptic case, as $km \approx 1230$ for both curves. Note that the values chosen for m in the elliptic case are selected solely to compare the relative efficiency of the two cases. More specifically, m is not chosen so that $\#E(\mathbb{F}_{2^m})$ has a large prime

Table 5 1 Experimental results - 950-bit security level

Case	Description	Running time (ms)
1	Genus 2 group order	1 89
2	Genus 2 η degenerate	1 71
3	Genus 2 η general	6 50
4	Genus 2 η_I degenerate	1 11
5	Genus 2 η_T general	3 60
6	Elliptic η_T	1 80

Table 5 2 Experimental results - 1230-bit security level

Case	Description	Running time (ms)
1	Genus 2 group order	2 69
2	Genus 2 η degenerate	2 60
3	Genus 2 η general	9 96
4	Genus 2 η_I degenerate	1 65
5	Genus 2 η_T general	5 48
6	Elliptic η_T	3 64

factor

Table 5 1 details the experimental results for the 950-bit security level, and Table 5 2 details the experimental results for the 1230-bit security level. The first case in each table is the time taken to compute the Tate pairing using the group order, as given in the previous chapter. The second case is the algorithm to compute the genus 2 η pairing using degenerate divisors, and the third case in each table is the algorithm to compute the genus 2 η pairing using general divisors. Similarly, the fourth case is an algorithm to compute the genus 2 η_T pairing using degenerate divisors, and the fifth case is an algorithm to compute the genus 2 η_T pairing using general divisors. The sixth case in each table is the equivalent timing using the elliptic curves E_d . All of the timings are given in milliseconds.

The first conclusion to be drawn from the results is that the genus 2 η pairing yields a running time approximately equivalent to that of using the group order to compute the Tate pairing. The genus 2 η pairing has a longer loop size than when the Tate pairing is computed using the group order. However, this result shows that this is compensated for by the simpler arithmetic of the η pairing inside the loop, as well as the cheaper final

exponentiation. The second conclusion is that the genus 2 η_T pairing is substantially more efficient than the genus 2 η pairing. This is not a surprising result, as the η_T pairing has a considerably shorter loop size than the η pairing. However, this is mitigated somewhat by the more complicated final exponentiation in the η_T case.

The third conclusion is that pairing implementation using degenerate divisors is considerably more efficient than the general case, for both of the genus 2 η and η_T pairings. This result confirms the findings of the previous chapter. However, the most surprising conclusion to be drawn from the experimental results is that the genus 2 η_T pairing on degenerate divisors outperforms the elliptic η_T pairing. This is the first time that it has been shown that a genus 2 pairing implementation can be faster than an equivalent elliptic curve implementation. This result is largely due to the smaller field sizes used in the genus 2 case, as the theoretical complexity of computing the genus 2 η_T pairing is larger than that of the elliptic η_T pairing. As far as we are aware, this timing for the genus 2 η_T pairing is also the fastest pairing implementation reported thus far in the literature.

All of the experiments were performed on our platform of a Pentium IV, which has a clock speed of 2.8 GHz, and which runs version 2.6.15 of the Linux kernel. The code is written in C/C++ and is compiled using version 4.0.3 of the GCC/G++ compiler suite. The efficient implementation of the finite field \mathbb{F}_{2^m} is taken from MIRACL 5.01. Recall that m was chosen in the genus 2 case so that arithmetic in \mathbb{F}_{2^m} can be performed in the 128-bit registers available to the Pentium IV. The SSE2 SIMD instruction set can then be used to multiply elements efficiently in $\mathbb{F}_{2^m}^*$. The elliptic curve implementation cannot use this optimisation as the field size is larger than 128 bits. We note that the timings provided are slightly different from the published timings, due to the need to remain consistent with experiments carried out in the previous chapter.

5.7 Conclusion

In this chapter, it was shown that it is possible to compute the Tate pairing extremely efficiently by using supersingular genus 2 curves over \mathbb{F}_{2^m} and the η_T pairing construct. Firstly, the η_T pairing was introduced as a generalisation and extension of the optimisations given by Duursma and Lee [23]. It was then shown how to apply the η_T pairing to the genus 2 case by using an order of $T = q = 2^{3m}$ in Miller's algorithm. This results in a loop of m iterations using the octupling formula given earlier. The subscript T is dropped for this case, and the pairing is known simply as the η pairing. Although the loop size is slightly longer than the $2m/3$ iterations of Miller's algorithm that is required when the group order is used, the η pairing compensates for this by having simpler arithmetic inside the loop, as well as a final exponentiation that is easily computed.

It was then shown how it is possible to compute the η pairing on supersingular elliptic curves in characteristic 2 without the final exponentiation to the power of $2^{2m} - 1$, at the cost of including the vertical line functions in Miller's algorithm. This technique does not offer any improvement as the final exponentiation is computed essentially as a multiplication and an inversion in $\mathbb{F}_{2^{4m}}$. It was also shown how a more efficient pairing calculation is possible in the genus 2 case by using a smaller value for T when computing the η_T pairing. This pairing implementation requires some extra arithmetic after the main loop, as well as a more complicated final exponentiation when compared to the η pairing. However, optimisations have been introduced to reduce these costs.

Experimental results were then detailed. It was shown that the genus 2 η pairing is approximately as efficient as the implementation of the Tate pairing that was given in the previous chapter. It was demonstrated that degenerate divisors yield a speed up over the more general case when computing both the η and η_T pairings. The experimental results also confirmed that the genus 2 η_T pairing is more efficient than the genus 2 η pairing. However, the results showed the surprising fact that the genus 2 η_T pairing is more efficient than the η_T pairing on supersingular elliptic curves over \mathbb{F}_{2^m} , using an equivalent level of security. This shows that not only are genus 2 curves competitive with elliptic curves in

terms of pairing implementation, but can in fact surpass them in certain circumstances

Kang and Park [57] show that some of the conditions given in this chapter on the formulation of the η_T pairing are unnecessary. Rather than using an automorphism γ , it is possible to simply use the multiplication by q . This results in a far simpler proof of the η_T pairing than that given in our paper [4]. Furthermore, it is shown that the only condition of the η_T pairing is that a supersingular curve be used. Kang and Park also show that the final exponentiation to T that must be performed when computing the η_T pairing can be replaced with an application of the q -th power Frobenius endomorphism. However, this does not result in any practical improvement, contrary to the authors' claims, as the main cost of the final exponentiation is the inversion that must be performed in the extension field.

In a paper by Lee et al. [72], our results on degenerate divisors are extended to general divisors in Mumford representation. More precisely, Lee et al. show how to extend our explicit formulae for computing the functions that are required in Miller's algorithm from the degenerate to the general case, and how to evaluate these functions at a general divisor. In a separate paper, Lee et al. [71] show how to use the η_I pairing construct to compute the Tate pairing using a hyperelliptic curve of genus 3. These curves are of the form $y^2 = x^7 - x + d$ over \mathbb{F}_{7^m} , where $d = \pm 1$, and have an embedding degree of $k = 14$. The curves that are used are part of the family of curves originally selected by Duursma and Lee for fast pairing computation. This is the first presentation of pairing computation in genus 3. However, the experimental results that are given are extremely inefficient compared to the results presented here for genus 2 curves.

In this chapter, experimental results for the genus 2 η and η_I pairings were given using a software implementation. However, we have also developed efficient implementations of these pairings in hardware. In Ronan et al. [95], we show how a dedicated parallel hardware implementation of the genus 2 η pairing yields an extremely efficient pairing computation. The fastest pairing implementation over $\mathbb{F}_{2^{103}}$ takes place in 749 μ -seconds. This paper was the first published paper on implementing genus 2 pairings in hardware, as well as being the first paper to implement a pairing in hardware using a finite field of characteristic

2 In a separate paper published as Ronan et al [94], we show how to implement the genus 2 η_T pairing in hardware. This implementation returned a time of 137 μ -seconds to compute a pairing over $\mathbb{F}_{2^{103}}$, which is extremely competitive with comparable elliptic curve implementations.

Chapter 6

Pairings on Supersingular Genus 2

Curves over \mathbb{F}_p

6.1 Introduction

Previous chapters described the implementation of the Tate pairing on a supersingular genus 2 curve over \mathbb{F}_{2^m} . This is the most logical choice of field for pairing implementation, as supersingular curves exist with an embedding degree of $k = 12$, which is the maximum possible for genus 2 curves over finite fields of arbitrary characteristic. However, it is also worthwhile to examine pairing implementation on genus 2 curves over a large prime field \mathbb{F}_p . Large prime fields are interesting as they are efficient to implement, and are resistant to sub-exponential time attacks on the DLP in \mathbb{F}_p^* . Theoretically, supersingular genus 2 curves exist over \mathbb{F}_p with an embedding degree of $k = 6$. However, only supersingular genus 2 curves with an embedding degree of $k = 4$ are known to the cryptographic community at this point.

In this chapter, it is shown how to efficiently implement the Tate pairing using a supersingular genus 2 curve over \mathbb{F}_p . Firstly, the curve is introduced, and formulae are given for doubling a divisor in $\text{Pic}_C^0(\mathbb{F}_p)$ and extracting the functions that are required for Miller's algorithm. These formulae are less expensive to compute than previous formulae given in

the literature. It is shown how to construct the field \mathbb{F}_{p^4} , and how to perform arithmetic efficiently in this field. It is shown how to exploit the distortion map to speed up the evaluation of the image divisor at the line functions, and how to compute the final exponentiation efficiently.

A new variant of Miller's algorithm is then described for hyperelliptic curves with an even embedding degree. This improvement eliminates divisions from Miller's algorithm even when the denominator elimination technique does not apply. This algorithm is useful in certain circumstances for curves of genus $g > 1$. A theoretical analysis is performed of the cost of computing the Tate pairing using the optimisations given in this chapter, and compared against previous work. Finally, experimental results are reported that establish new benchmarks for pairing implementation on genus 2 curves over large prime fields, and the chapter is concluded.

This chapter contains joint work with Michael Scott, which has been accepted for publication in the proceedings of Selected Areas in Cryptography, 2006. A preprint is available at the ePrint archive as O hEigartaigh and Scott [47]. Some of the work on eliminating divisions in Miller's algorithm previously appeared in a short paper at the ePrint archive as O hEigartaigh [46], and was presented at the rump session of the ECC 2005 conference. We note that this idea was derived independently by Kobayashi et al [62] for the case of elliptic curves.

6.2 The Curve

The first task is to select a supersingular genus 2 curve over \mathbb{F}_p with an embedding degree that is suitable for pairing based cryptography. Choe et al [16] examine genus 2 curves given by an equation of the form

$$y^2 = x^5 + a, \quad a \in \mathbb{F}_p^*$$

Choi et al show that this curve is supersingular whenever $p \not\equiv 1 \pmod{5}$. To determine the embedding degree of the curve for the other congruence conditions on p , it is required to compute the group order $\#\text{Pic}_C^0(\mathbb{F}_p)$. Recall that the characteristic polynomial of the Frobenius endomorphism is given in the genus 2 case as

$$\chi_C(T) = T^4 + a_1T^3 + a_2T^2 + a_1pT + p^2,$$

where $a_1 = \#C(\mathbb{F}_p) - 1 - p$ and $a_2 = (\#C(\mathbb{F}_{p^2}) - 1 - p^2 + a_1^2)/2$. Once a_1 and a_2 have been derived, the group order is computed as $\chi_C(1) = \#\text{Pic}_C^0(\mathbb{F}_p)$. Choi et al show that when $p \equiv 2, 3 \pmod{5}$, then $\#C(\mathbb{F}_p) = p + 1$ and $\#C(\mathbb{F}_{p^2}) = p^2 + 1$, and hence $a_1 = a_2 = 0$. Therefore, the group order is given as $\chi_C(1) = p^2 + 1$. As $(p^4 - 1)$ is the smallest term into which $(p^2 + 1)$ divides evenly, then the embedding degree of the curve is $k = 4$. When $p \equiv 4 \pmod{5}$, Choi et al show that the group order is equal to $\#\text{Pic}_C^0(\mathbb{F}_p) = (p + 1)(p + 1)$, and that the embedding degree of the curve is bounded by $k = 2$ as a result.

Therefore, the curve that is used for pairing implementation in this chapter is given as

$$C: y^2 = x^5 + a, \quad a \in \mathbb{F}_p^*, \quad p \equiv 2, 3 \pmod{5}$$

Rather than select an arbitrary value in \mathbb{F}_p^* for a , we take $a = 1$ for convenience. This curve was used by Choi and Lee [17] to implement the Tate pairing. They define the distortion map ψ that maps elements in $C(\mathbb{F}_p)$ to $C(\mathbb{F}_{p^4})$ as

$$\psi(x, y) = (\zeta_5 x, y),$$

where ζ_5 is a primitive 5th root of unity in \mathbb{F}_{p^4} . Note that ζ_5 maps the x -coordinate to \mathbb{F}_{p^4} , and hence the denominator elimination technique does not apply. To see that ψ is an

endomorphism on C , observe that

$$y^2 = (\zeta_5 x)^5 + a = \zeta^5 x^5 + a = x^5 + a$$

It remains to examine which order to use in Miller's algorithm. One option is to use the group order $\#\text{Pic}_C^0(\mathbb{F}_p) = p^2 + 1$. This has the advantage of having a trivial final exponentiation of $p^2 - 1$, which can be computed essentially as a multiplication and an inversion in \mathbb{F}_{p^4} . However, as p is a large prime number, then the number of loop iterations in Miller's algorithm, $\log_2(p^2 + 1)$, will also be large. The second option is to use a prime-order subgroup. The advantage of this approach is that the prime order n of the subgroup can be chosen to have a low Hamming weight, which results in a smaller number of additions in Miller's algorithm. If n is chosen to be considerably smaller than $\#\text{Pic}_C^0(\mathbb{F}_p)$, then the number of loop iterations will also be small. Therefore, it is better to use a prime subgroup order rather than the full group order in Miller's algorithm.

There are a number of criteria on the selection of the prime subgroup order n and the large prime p . Firstly, n must be large enough to resist any generic attacks on the DLP in $\text{Pic}_C^0(\mathbb{F}_p)[n]$. Secondly, $\mathbb{F}_{p^4}^*$ must be large enough to resist any sub-exponential time attacks on the DLP. Choosing suitable values for n and p is a tricky problem, due to the wide range of algorithms available for solving the DLP in $\text{Pic}_C^0(\mathbb{F}_p)[n]$ and $\mathbb{F}_{p^4}^*$. We follow the parameters defined by Lenstra and Verheul [74], which were used by Scott [105] to implement the Weil pairing. These parameters are defined as $(160/1024)$, $(192/2048)$ and $(224/4096)$, where the first number in each term is $\log_2(n)$, and the second number is $\log_2(p^k)$. For a thorough comparison of security levels, the reader can consult Galbraith et al [35].

The security levels given above detail the number of bits that are required for both n and p . The first task is to choose concrete values for the prime subgroup order n . Recall that Barreto et al [5] explore the use of Solinas primes [109] when computing the Tate pairing. These are prime numbers of the form $n = 2^\alpha \pm 2^\beta \pm 1$. When a Solinas prime

Table 6.1 The prime subgroup order n for each security level

Case	$\log_2(n)$	n
1	160	$n = 2^{159} + 2^{17} + 1$
2	192	$n = 2^{191} + 2^2 + 1$
3	224	$n = 2^{223} + 2^{13} + 1$

is used in Miller's algorithm only two additions are required. Also, Duursma and Lee [23] show that the final addition can be skipped when the denominator elimination idea applies, as it corresponds to a vertical line function. Therefore, a Solinas prime n is used as the subgroup order, where $\log_2(n) \approx 160, 192$ and 224 , for each of the three security levels defined above. The values for n that are chosen are given in Table 6.1.

The next step is to choose the large prime p . As the embedding degree of the curve is $k = 4$, p must be chosen so that $\log_2(p) \approx 256, 512$ and 1024 . However, a condition on p is that the subgroup order n must divide the group order $\#\text{Pic}_C^0(\mathbb{F}_p) = p^2 + 1$. The probability of a randomly chosen prime number p of the correct number of bits satisfying this condition is negligible. Therefore, a method must be given to construct p . Firstly, note that the condition on p can be rephrased as $p^2 + 1 \equiv 0 \pmod{n}$, and hence $p \equiv \sqrt{-1} \pmod{n}$. It is a well known fact that -1 is a quadratic residue modulo n if and only if $n \equiv 1 \pmod{4}$. All of the values selected for n in Table 6.1 satisfy this property.

Therefore, the method to compute p is as follows. Let n be the prime subgroup order, such that $n \equiv 1 \pmod{4}$, and let $t = \sqrt{-1} \pmod{n}$. Choose a random value w , such that $p = wn + t$ has the desired number of bits. Then continually add n to this value, until p is a prime number with the required congruence conditions. This method converges quickly on a suitable prime number p . The actual values obtained for p for each of the three security levels will be given later in this chapter. Note that n^2 should not divide $\#\text{Pic}_C^0(\mathbb{F}_p)$, for reasons outlined in Chapter 2.

As the subgroup order n has a very low Hamming weight, the number of additions to be performed in Miller's algorithm is negligible. However, a doubling of an element in $\text{Pic}_C^0(\mathbb{F}_p)$ must take place at each iteration of the loop. Therefore, it is worth examining

Table 6 2 Comparison of the cost of doubling in $\text{Pic}_C^0(\mathbb{F}_p)$

Origin	Doubling	$l(x)$
Miyamoto et al [85]	1, 23M, 4S	3M
Lange [70]	1, 22M, 5S	3M
Choi and Lee [17]	1, 23M, 5S	no cost
Our work	1, 22M, 4S	no cost

how this operation can be optimised Lange [70] gives explicit formulae for doubling a divisor with a cost of 1 inversion, 22 multiplications and 5 squarings in \mathbb{F}_p (in the overwhelmingly common case) However, these formulae are designed to be used in the context of scalar multiplication, and do not explicitly calculate all of the functions that are required in Miller’s algorithm The formulae can easily be modified to extract the functions that are required, at the cost of 3 extra multiplications

Choi and Lee [17] modify Lange’s formulae for doubling a general divisor to reduce the cost of calculating the functions that are required in Miller’s algorithm The formulae they present cost 1 inversion, 23 multiplications and 5 squarings in \mathbb{F}_p , thereby saving 2 multiplications over the previous approach However, it is possible to improve on these formulae In Table 6 3, we give formulae to double a divisor $[u_1, v_1]$, in the overwhelmingly common case that the degree of u_1 is 2, and $\gcd(u_1, 2v_1) = 1$ The cost of these formulae is 1 inversion, 22 multiplications and 4 squarings (the multiplication is saved in step 8) Note that as the characteristic of \mathbb{F}_p is odd, the $h(x)$ polynomial is assumed to be zero (where $h(x)$ is from the equation of the curve $y^2 + h(x)y = f(x)$)

We believe that the formulae in Table 6 3 are optimal, as they have the same computational cost as simply doubling a divisor as given by Lange [70] (in fact a squaring is saved over these formulae) In other words, calculating the functions that are required in Miller’s algorithm does not cost anything extra over the cost of doubling a divisor Table 6 2 summarises the computational cost of doubling a general divisor

Table 6.3 Formulae for doubling for the curve $C: y^2 = x^5 + a$

Input	$D_1 = [u_1, v_1]$ where $u_1 = x^2 + u_{11}x + u_{10}, v_1 = v_{11}x + v_{10}$	
Output	$D_3 = [u_3, v_3], l(x)$ such that $[2]D_1 = D_3 + ((y - l(x))/u_3(x))$	
Step	Expression	Cost
1	Compute $\tilde{v}_1 \equiv (2v_1) \pmod{u_1} = \tilde{v}_{11}x + \tilde{v}_{10}$ $\tilde{v}_{11} = 2v_{11}, \tilde{v}_{10} = 2v_{10}$	
2	Compute $r = \text{res}(u_1, \tilde{v}_1)$ $w_0 = v_{11}^2, w_1 = u_{11}^2, w_2 = 4w_0, w_3 = u_{11}\tilde{v}_{11},$ $r = u_{10}w_2 + \tilde{v}_{10}(w_1 - w_3)$	2S + 3M
3	Compute almost inverse of $inv' = r(2v_1)^{-1} \pmod{u_1}$ $inv'_1 = -\tilde{v}_{11}, inv'_0 = \tilde{v}_{10} - w_3$	
4	Compute $k' = \frac{f - v_1^2}{u_1} \pmod{u_1} = k'_1x + k'_0$ $w_3 = w_1, w_4 = 2u_{10}, k'_1 = 2w_1 + w_3 - w_4$ $k'_0 = u_{11}(2w_4 - w_3) - w_0$	1M
5	Compute $s' = k' inv' \pmod{u_1}$ $w_0 = k'_0 inv'_0, w_1 = k'_1 inv'_1$ $s'_1 = \tilde{v}_{10}k'_1 - \tilde{v}_{11}k'_0, s'_0 = w_0 - u_{10}w_1$ If $s'_1 = 0$ then goto step 6'	5M
6	Compute $s = s_1x + s_0$ and s_1^{-1} $w_1 = (rs'_1)^{-1}, w_2 = s'_1w_1, w_3 = r^2w_1,$ $s_1 = s'_1w_2, s_0 = s'_0w_2$	1I, 1S, 5M
7	Compute $l(r) = su_1 + v_1 = s_1r^3 + l_2r^2 + l_1r + l_0$ $l_2 = s_1u_{11} + s_0, l_0 = s_0u_{10} + v_{10}$ $l_1 = (s_1 + s_0)(u_{11} + u_{10}) - s_1u_{11} - s_0u_{10} + v_{11}$	3M
8	Compute $u' = \text{monic}(\frac{f - l^2}{u_1^2}) = x^2 + u_{31}x + u_{30}$ $u_{30} = w_3(\tilde{v}_{11} + w_3(2u_{11} + s_0^2))$ $u_{31} = 2s_0 - w_3$	1S + 2M
9	Compute $v_3 = -l \pmod{u_3} = v_{31}x + v_{30}$ $w_1 = u_{31}, u_{31} = w_3u_{31}, w_3 = l_2 - w_1, w_3 = u_{30}w_2$ $v_{31} = (u_{31} + u_{30})(w_2 + s_1) - w_3 - w_1 - l_1, v_{30} = w_3 - l_0$	3M
		1I, 4S, 22M
6'	Compute $l(x) = s_0u_1 + v_1$ $inv = 1/r, s_0 = s'_0 inv, l_1 = s_0u_{11} + v_{11}, l_0 = s_0u_{10} + v_{10}$	1I + 3M
7'	Compute $u_3 = \text{monic}(\frac{f - l^2}{u_1^2}) = x + u_{30}$ $u_{30} = -2u_{11} - s_0^2$	1S
8'	Compute $v_3 = -l \pmod{u_3} = v_{30}$ $v_{30} = u_{30}(l_1 - u_{30}s_0) - l_0$	2M
		1I, 3S, 14M

6.3 Curve Arithmetic

In this section, various techniques are described that enable the efficient implementation of the Tate pairing on a supersingular genus 2 curve over \mathbb{F}_p

6.3.1 Finite field arithmetic

As the embedding degree of the curve C is $k = 4$, it must be shown how to construct the finite field \mathbb{F}_{p^4} . For reasons outlined in Chapter 2, a polynomial basis representation is used rather than a normal basis representation. Rather than construct \mathbb{F}_{p^4} as a quartic extension of \mathbb{F}_p , the field \mathbb{F}_{p^2} is first constructed using an irreducible quadratic polynomial defined over \mathbb{F}_p . Then the field \mathbb{F}_{p^4} is defined as a quadratic extension of \mathbb{F}_{p^2} , by using an irreducible quadratic polynomial defined over \mathbb{F}_{p^2} . The advantage of this approach is that it is easier to optimise the arithmetic in \mathbb{F}_{p^2} than in a quartic extension of \mathbb{F}_p . These techniques can then be reused with minor modifications to optimise the arithmetic in \mathbb{F}_{p^4} .

The first task is to give irreducible polynomials that define the fields \mathbb{F}_{p^2} and \mathbb{F}_{p^4} . An irreducible binomial $x^2 - \beta$ is used to define the extension field \mathbb{F}_{p^2} , where β is a quadratic non-residue in \mathbb{F}_p . The quadratic extension of \mathbb{F}_{p^2} can then be constructed by adjoining the quartic-root of β . It might seem like a good idea to choose $\beta = -1$. It is well known that -1 is a quadratic non-residue with respect to p if and only if $p \equiv 3 \pmod{4}$. However, a quartic root of -1 exists in \mathbb{F}_{p^2} , and so it is not possible to build a quadratic extension of \mathbb{F}_{p^2} using $x^4 + 1$. A second choice for β is $\beta = -2$. If the prime p is congruent to $5 \pmod{8}$, then -2 is a quadratic non-residue with respect to p . This value for β permits the construction of \mathbb{F}_{p^4} as a quadratic extension of \mathbb{F}_{p^2} . Another advantage of using a prime $p \equiv 5 \pmod{8}$ is that a simple formula exists to compute square roots modulo p , which is required for generating random points on the curve.

Elements of the field \mathbb{F}_{p^2} are then represented as $(a + b\sqrt{\beta})$, where $a, b \in \mathbb{F}_p$, and elements of the field \mathbb{F}_{p^4} are represented as $(c + d\sqrt[4]{\beta})$, where $c, d \in \mathbb{F}_{p^2}$. Addition and subtraction in \mathbb{F}_{p^2} and \mathbb{F}_{p^4} are relatively cheap to compute. However, it is worth examining

how to optimise multiplication and squaring in these fields, as they are expensive operations to compute. Let M and S be a multiplication and squaring respectively in \mathbb{F}_p . The schoolbook method to multiply two elements in \mathbb{F}_{p^2} costs $4M$. However, it is possible to do better by using the Karatsuba technique in the following manner

$$\begin{aligned}(a + b\sqrt{\beta})(x + y\sqrt{\beta}) &= (ax - 2by + (ay + bx)\sqrt{\beta}) \\ &= (ax - 2by + ((a + b)(x + y) - ax - by)\sqrt{\beta})\end{aligned}$$

This costs only $3M$. Similarly, a multiplication of two elements in \mathbb{F}_{p^4} costs 3 multiplications in \mathbb{F}_{p^2} , and hence $9M$ in total. The next task is to examine squaring in \mathbb{F}_{p^2} and \mathbb{F}_{p^4} . It is commonly assumed that a squaring is computationally equivalent to a multiplication when estimating the cost of pairing operations (e.g. see Koblitz and Menezes [65]). As detailed in Chapter 2, this is a reasonably valid assumption in \mathbb{F}_p . However, it is possible to optimise squaring routines in extension fields so that a squaring is considerably less expensive to compute than a multiplication. The schoolbook method to square an element in \mathbb{F}_{p^2} costs $2S + M$. However, it is better to exploit the Karatsuba technique again as

$$\begin{aligned}(a + b\sqrt{\beta})^2 &= (a^2 - 2b^2 + 2ab\sqrt{\beta}) \\ &= ((a + b)(a - 2b) + ab + 2ab\sqrt{\beta})\end{aligned}$$

This costs $2M$, which is M cheaper than the cost of a general multiplication in \mathbb{F}_{p^2} . Similarly, squaring an element in \mathbb{F}_{p^4} costs 2 multiplications in \mathbb{F}_{p^2} , and thus $6M$ in total. This is considerably cheaper than the $9M$ required for a general multiplication in \mathbb{F}_{p^4} . As the accumulating variable $f \in \mathbb{F}_{p^4}$ is squared at each iteration of Miller's algorithm, using this optimised squaring method is a substantial improvement over using a general multiplication routine.

6.3.2 Evaluating the line functions

The use of degenerate divisors was previously discussed in Chapter 4 for the characteristic 2 case. Recall that for a genus 2 curve, a degenerate divisor $D = (P) - (\infty) \in \text{Pic}_C^0(\mathbb{F}_p)$ has a single finite point P in the support. This is in contrast to the more general reduced divisor $D = (P) + (Q) - 2(\infty)$, with two finite points in the support. In Chapter 4, the first argument to Miller's algorithm was defined to be a degenerate divisor, as an automorphism on the curve was used to keep the divisor in its special shape. However, there is no advantage to be gained from doing this for the genus 2 curve under consideration in this chapter, as the first doubling will turn the degenerate divisor into a general divisor. Therefore, a general divisor is used as the first input to Miller's algorithm for all of the pairing implementations in this chapter.

However, a distinct advantage still exists in using a degenerate divisor as the second input to Miller's algorithm. Each time a doubling or addition is performed in Miller's algorithm, a function is evaluated at the image divisor. If the image divisor is a general divisor, then the evaluation takes place using either the two finite points in the support of the divisor, or the Mumford representation of the divisor. However, if a degenerate divisor $D = (P) - (\infty)$ is used as the image divisor, it is possible to evaluate at the finite point P . This approach yields a modest speedup over using general divisors. Frey and Lange [29] discuss in detail when it is permissible to choose a degenerate divisor as the second argument to Miller's algorithm.

Here it is assumed that the line functions in Miller's algorithm are evaluated at a degenerate divisor $D_2 = (\psi(Q)) - (\infty)$, where $\psi(Q)$ is the point in \mathbb{F}_{p^4} that results from the application of the distortion map ψ to a point $Q \in C(\mathbb{F}_p)$. At each iteration of the loop, the iterating divisor D_1 is doubled using Cantor's composition algorithm to get the divisor $[2]D_1$. A reduced divisor D_3 equivalent to $[2]D_1$ is then obtained by using Cantor's reduction algorithm, such that $[2]D_1 = D_3 + ((y - l(x))/u_3(x))$, where the functions $l(x)$ and $u_3(x)$ are extracted from the composition and reduction process. These functions are then evaluated at $\psi(Q)$, an inversion is performed on $u_3(x)$, and both functions are multiplied

by the accumulating variable

Firstly, the evaluation of $\psi(Q) = (x_{\psi(Q)}, y_{\psi(Q)})$ at the function $y - l(x)$ is examined. The function $l(x)$ that is extracted from the composition process is given as $l(x) = (x^3 s_1 + x^2 l_2 + x l_1 + l_0)$. Note that s_1, l_2, l_1 and l_0 are defined in \mathbb{F}_p , as the iterating divisor is also defined over \mathbb{F}_p . The values $x_{\psi(Q)}^2$ and $x_{\psi(Q)}^3$ can be precomputed, and so evaluating this function at $\psi(Q)$ has a cost of 12 multiplications in \mathbb{F}_p . However, it is possible to save some multiplications over this by manipulating the image point $\psi(Q)$. Rather than explicitly calculate the distortion map on Q , it is possible to build the distortion map into the function evaluation. Let $Q = (x_Q, y_Q) \in C(\mathbb{F}_p)$. Recall that $\psi(Q) = (\zeta_5 x_Q, y_Q) \in C(\mathbb{F}_{p^4})$, where ζ_5 is a primitive 5th root of unity in \mathbb{F}_{p^4} . Then $-l(x)$ is written as

$$-l(x) = -((x_Q \zeta_5)^3 s_1 + (x_Q \zeta_5)^2 l_2 + (x_Q \zeta_5) l_1 + l_0)$$

Two multiplications can be saved in this function evaluation by examining the relation between certain powers of ζ_5 . If ζ_n is a primitive n^{th} root of unity in a field K , then its conjugates over the prime subfield K_0 of K are also primitive n^{th} roots of unity [92]. Also, ζ_n^a is a primitive n^{th} root of unity if and only if a and n are co-prime. Applying this to ζ_5 means that the third power of ζ_5 is related to the second power by conjugation, in other words $\zeta_5^3 = \overline{\zeta_5^2}$.

Note that $-l(x)$ as defined above can be written in the form $a + b\zeta_5 + c\zeta_5^2 + d\zeta_5^3$, where $b = -x_Q l_1$, $c = -x_Q^2 l_2$ and $d = -x_Q^3 s_1$. Let $\zeta_5^2 = (m + n\sqrt[4]{\beta})$ where $m, n \in \mathbb{F}_{p^2}$. Then it is possible to compute the function $a + b\zeta_5 + c\zeta_5^2 + d\zeta_5^3$ as $a + b\zeta_5 + ((c + d)m + (c - d)n\sqrt[4]{\beta})$. Computing c and d takes only 2 multiplications in \mathbb{F}_p (with a precomputation of 1 squaring and 1 multiplication). Computing $(c + d)m$ and $(c - d)n$ takes 4 multiplications, with a precomputation of 6 multiplications. Computing $b\zeta_5^5$ takes 4 multiplications, with a precomputation of 4 multiplications. Therefore, the total multiplication count in evaluating the function is 10 multiplications, a saving of two multiplications, with a precomputation of 11 multiplications and 1 squaring.

Evaluating the image point $\psi(Q)$ at the function $u_3(x) = x^2 + u_{31}x + u_{30}$ costs 8 multiplications in \mathbb{F}_p , by precomputing $(x_Q \zeta_5)$ and $(x_Q \zeta_5)^2$. This precomputation costs 6 multiplications, by reusing some of the precomputation needed to compute the line function $y - l(x)$. Therefore, the total cost of evaluating both functions at $\psi(Q)$ is given as 18 multiplications in \mathbb{F}_p per iteration of the loop, with a precomputation of 1 squaring and 17 multiplications in \mathbb{F}_p .

6.3.3 The final exponentiation

An exponentiation must be performed on the output of Miller's algorithm to compute the (reduced) Tate pairing. For a genus 2 curve with an embedding degree of $k = 4$, this exponentiation is to the power of $(p^4 - 1)/n$. As the embedding degree of the curve is even, it is possible to extract the factor $(p^2 - 1)$ from the final exponentiation. Exponentiating to this power can be trivially computed with a conjugation with respect to \mathbb{F}_{p^2} , a multiplication and an inversion in \mathbb{F}_{p^4} . Using the basis described previously, a conjugation with respect to \mathbb{F}_{p^2} is implemented as $\bar{x} = (a - b\sqrt[4]{\beta})$, for an element $x = (a + b\sqrt[4]{\beta}) \in \mathbb{F}_{p^4}$. The remaining exponentiation to $(p^2 + 1)/n$ is an expensive operation to compute as it cannot be simplified further. As detailed in Chapter 3, there are two techniques that are used to compute this exponentiation efficiently.

The first approach is to use Lucas exponentiation, as proposed by Scott and Barreto [106]. This method was detailed in Chapter 3. The alternative strategy is due to Hu et al. [50] and Granger et al. [42]. The remaining exponentiation is given as $x^{(p^2+1)/n}$ for an element $x \in \mathbb{F}_{p^4}$. Firstly, note that $(p^2 + 1)/n = a_1 p + a_0$, where $a_1 = (p^2 + 1)/(pn)$ and $a_0 = ((p^2 + 1)/n) \bmod p$. The main idea is to exploit the fact that exponentiating an element in \mathbb{F}_{p^4} to the power of p can be trivially computed. Therefore, the exponentiation can be performed by precomputing a_1 and a_0 , and by evaluating $(x^p)^{a_1} x^{a_0}$. Granger et al. show how the technique of multi-exponentiation can be exploited to compute this term. Essentially, the idea behind multi-exponentiation is to use a single square-and-multiply algorithm to compute both exponentiations simultaneously. This idea is also known as Shamir's trick.

Table 6.4 Experimental results for the final exponentiation

Security level	Multi-exponentiation (ms)	Lucas exponentiation (ms)
(160/1024)	2.6	1.3
(192/2048)	12.5	7.5
(224/4096)	72.5	48.5

Granger et al. give theoretical results that show that it is faster to use the Lucas sequence approach for curves with a low embedding degree. Scott [105] provides experimental evidence to verify this, by stating that Lucas exponentiation is better only when the embedding degree of the curve is $k \leq 4$. In Table 6.4, timings are given on our platform of a Pentium IV, 2.8 GHz, to illustrate the performance of the Lucas exponentiation approach versus the multi-exponentiation approach to compute $\alpha^{(p^2+1)/n}$ for the three security levels of our curve. As can be seen, the Lucas sequence approach is superior for all three levels.

6.4 Computing the Tate Pairing

In this section, a new variant of Miller's algorithm to compute the Tate pairing is described, and is compared against the denominator elimination technique for the genus 2 curve in question. A theoretical analysis is also performed on the cost of computing the Tate pairing using our optimisations.

6.4.1 Modifying Miller's algorithm

Recall that Miller's algorithm as originally described involves performing an inversion in \mathbb{F}_{p^k} at each iteration of the loop. Field inversion is an expensive operation to compute, particularly so in the extension field \mathbb{F}_{p^k} . Galbraith et al. [31] introduce a variant of Miller's algorithm, which removes the need to perform an inversion at each iteration of the loop. The basic idea of Galbraith et al. is to postpone performing the inversion until after the loop. To achieve this, two variables are used in the loop, which effectively replaces an inversion with a squaring at each loop iteration. The algorithm of Galbraith et al. to compute the

Tate pairing $\langle D_1, D_2 \rangle_n$, where $D_1 \in \text{Pic}_C^0(\mathbb{F}_p)$ and $D_2 \in \text{Pic}_C^0(\mathbb{F}_{p^k})$, is presented in Algorithm 11

Algorithm 11 Miller's algorithm to compute the Tate pairing, as per Galbraith et al [31]

INPUT $D_1 \in \text{Pic}_C^0(\mathbb{F}_p), D_2 \in \text{Pic}_C^0(\mathbb{F}_{p^k})$, where D_1 has order n

OUTPUT $\langle D_1, D_2 \rangle_n^{(q^k-1)/n}$

```

1  $f_c \leftarrow 1, f_d \leftarrow 1$ 
2  $T \leftarrow D_1$ 
3 for  $i \leftarrow \lfloor \log_2(n) \rfloor - 1$  downto 0 do
4    $\triangleright$  Compute  $T' = (2)T - (c/d)$ 
5    $T \leftarrow [2]T$ 
6    $f_c \leftarrow f_c^2 \cdot c(D_2), f_d \leftarrow f_d^2 \cdot d(D_2)$ 
7   if  $n_i = 1$  then
8      $\triangleright$  Compute  $T' = T + D_1 - (c/d)$ 
9      $T \leftarrow T + D_1$ 
10     $f_c \leftarrow f_c \cdot c(D_2), f_d \leftarrow f_d \cdot d(D_2)$ 
11   end if
12 end for
13  $f \leftarrow f_c/f_d$ 
14  $f \leftarrow f^{(p^k-1)/n}$ 
15 Return  $f$ 

```

An important improvement on the approach of Galbraith et al is the denominator elimination technique of Barreto et al [5]. In Algorithm 11, the iterating divisor D_1 is an element of the group $\text{Pic}_C^0(\mathbb{F}_p)$, rather than $\text{Pic}_C^0(\mathbb{F}_{p^k})$. As a result, all of the coefficients of the line functions are also defined over \mathbb{F}_p , as they are extracted from the addition process on D_1 . Rather than defining the image divisor D_2 to be a general element of $\text{Pic}_C^0(\mathbb{F}_{p^k})$, let the x -coordinates of all of the finite points in the support of D_2 be defined over some subfield of \mathbb{F}_{p^k} . In this case the denominator function, or the f_d variable in Algorithm 11, will also be defined over a subfield of \mathbb{F}_{p^k} . However, the exponentiation to $(p^k/2 - 1)$ which takes place as part of the final exponentiation eliminates any function value that is defined over $\mathbb{F}_{p^{k/2}}$. Therefore, there is no need to compute the f_d variable in Algorithm 11.

Several techniques are used to implement the denominator elimination technique in practice. The first method uses a distortion map to map elements of $\text{Pic}_C^0(\mathbb{F}_p)$ to $\text{Pic}_C^0(\mathbb{F}_{p^k})$, where C is a supersingular curve. Some distortion maps map the x -coordinates to a subfield

of \mathbb{F}_{p^k} by definition, and so denominator elimination applies when such a divisor is used as the second argument to Miller's algorithm. The second method uses quadratic twists of elliptic curves, as covered in Chapter 2. The third method defines the second argument D_2 to Miller's algorithm to be a trace-zero divisor. Let $D' \in \text{Pic}_C^0(\mathbb{F}_{p^k})$ be a general divisor. Then a trace-zero divisor is computed as $D_2 = D' - D'^{p^{k/2}}$, which supports the denominator elimination technique.

However, suppose that a degenerate divisor is used as the second argument to Miller's algorithm for a hyperelliptic curve of genus $g > 1$. Then if this technique is used in order to implement denominator elimination, it will increase the weight of the image divisor. Recall that a degenerate divisor D_2 in the genus 2 context has a single finite point in the support. Applying the trace-zero map to D_2 results in a more general divisor with two finite points in the support. For elliptic curves the divisor class group is isomorphic to the group of points, and thus any non-trivial class has exactly one finite point in the support. Therefore, to use the denominator elimination technique in the genus 2 case, the functions in Miller's algorithm must be evaluated at two points, rather than at a single point. This reduces the efficiency of denominator elimination.

However, we present an alternative way to proceed, by introducing a new variant of Miller's algorithm to compute the Tate pairing. A prerequisite for this algorithm is that the embedding degree of the curve must be even, a condition shared by the denominator elimination technique. Firstly, it is assumed that the finite extension field \mathbb{F}_{p^k} is represented as a quadratic extension of $\mathbb{F}_{p^{k/2}}$. It is well known that once an element $x = (a + b\sqrt{\beta}) \in \mathbb{F}_{p^k}$ is raised to the power of $p^{k/2} - 1$, then it is possible to replace an inversion with a conjugation, i.e. $(\frac{1}{x})^{p^{k/2}-1} = (\bar{x})^{p^{k/2}-1}$. To see why this is so note that

$$\left(\frac{1}{x}\right)^{p^{k/2}-1} = \frac{1}{(a + b\sqrt{\beta})^{p^{k/2}-1}} = \frac{(a + b\sqrt{\beta})}{(a - b\sqrt{\beta})}$$

Similarly

$$\bar{x}^{p^{k/2}-1} = \overline{(a + b\sqrt{\beta})^{p^{k/2}-1}} = (a - b\sqrt{\beta})^{p^{k/2}-1} = \frac{(a + b\sqrt{\beta})}{(a - b\sqrt{\beta})}$$

This effectively replaces an expensive operation with one that is free to compute

This technique is exploited by Scott [105] to compute the Weil pairing. Scott proposes exponentiating the pairing value to the power of $p^{k/2} - 1$, which means that the inversion in the Miller loop can be replaced with a conjugation. However, no one has previously observed that it is possible to use this idea to compute the Tate pairing, without requiring any additional exponentiation. The final exponentiation required to compute the reduced Tate pairing includes the factor $(p^{k/2} - 1)$, as $(p^k - 1)/n = (p^{k/2} - 1)(p^{k/2} + 1)/n$. Therefore, as the output of the loop is implicitly raised to the power of $(p^{k/2} - 1)$, there is no need for the strategy of Galbraith et al. of using two variables to eliminate inversion, as the inversion in the main loop can be replaced by a conjugation. The new algorithm is described in Algorithm 12 for the hyperelliptic case. As the variable f_d is eliminated from the pairing calculation, a squaring is saved in \mathbb{F}_{p^k} at each iteration of the loop, when compared to Algorithm 11.

Algorithm 12 An improved algorithm to compute the Tate Pairing

INPUT $D_1 \in \text{Pic}_C^0(\mathbb{F}_p)$, $D_2 \in \text{Pic}_C^0(\mathbb{F}_{p^k})$, where D_1 has order n

OUTPUT $\langle D_1, D_2 \rangle_n^{(p^k-1)/n}$

```

1  $f \leftarrow 1$ 
2  $T \leftarrow D_1$ 
3 for  $i \leftarrow \lfloor \log_2(n) \rfloor - 1$  downto 0 do
4    $\triangleright$  Compute  $T' = (2)T - (c/d)$ 
5    $T \leftarrow [2]T$ 
6    $f \leftarrow f^2 \cdot \frac{c(D_2)}{d(D_2)}$ 
7   if  $n_i = 1$  then
8      $\triangleright$  Compute  $T' = T + D_1 - (c/d)$ 
9      $T \leftarrow T + D_1$ 
10     $f \leftarrow f \cdot \frac{c(D_2)}{d(D_2)}$ 
11   end if
12 end for
13  $f \leftarrow f^{(p^k-1)/n}$ 
14 Return  $f$ 

```

Table 6 5 Complexity of function calculation per iteration in Miller’s Algorithm

Case	Description	Complexity
1	Original Approach	1, 2M, S
2	Two-variable Approach	2M, 2S
3	Algorithm 12	2M, S
4	Denominator Elimination	M, S

Algorithm 12 is still not as efficient as the denominator elimination technique, which saves a multiplication over this again at each iteration. However, Algorithm 12 is a slightly more efficient technique to compute the Tate pairing on a hyperelliptic curve of genus $g > 1$, when using a distortion map that does not admit denominator elimination directly. The reason for this is that the denominator elimination algorithm consists of two evaluations at the line function at each iteration (or one evaluation of a more complicated form if Mumford representation is used). Algorithm 12 consists of one evaluation at the line function, and one evaluation at the vertical line function, which requires less computation to evaluate than the line function. Algorithm 12 is also less restrictive than using denominator elimination, as it places no conditions on the form of the image divisor. In fact, for an arbitrary image divisor D_2 which is fully defined over \mathbb{F}_{p^k} , it is the most efficient algorithm to compute the Tate pairing in the literature. Table 6 5 illustrates the complexity of the different algorithms in more detail.

6 4 2 Using denominator elimination

Algorithm 12 is more efficient than the denominator elimination technique, assuming the use of a distortion map that does not give denominator elimination directly, and that the image divisor is a degenerate divisor. However, it is possible to reduce the performance gap by using customized multiplication routines, as detailed in this section. Given a degenerate divisor $D = \psi(Q) - (\infty)$, where $\psi(Q) = (x, y) \in C(\mathbb{F}_{p^4})$, the transformation $R = \psi(Q) - \psi(Q)^{p^2}$ yields an effective trace-zero divisor R that is suitable for use with the denominator elimination technique. This transformation can be easily computed, by using

the fact that exponentiating an element in \mathbb{F}_{p^4} to the power of p^2 is equivalent to a simple conjugation with respect to the quadratic subfield, and so

$$-\psi(Q)^{p^2} = -(\bar{x}, \bar{y}) = (\bar{x}, -y)$$

Therefore, rather than compute R using Cantor composition and reduction, note that R can be constructed as $R = (r, y) + (\bar{r}, -y) - 2(\infty)$. The fact that the two finite points in the support of R are similar can be exploited in Miller's algorithm. Let $Q = (r_Q, y_Q) \in C(\mathbb{F}_p)$. Evaluating $\psi(Q)$ at the line function $y - l(r)$ gives

$$y_Q - ((x_Q \zeta_5)^3 s_1 + (x_Q \zeta_5)^2 l_2 + (x_Q \zeta_5) l_1 - l_0)$$

Rather than evaluate the line function at $-\psi(Q)^{p^2}$ separately, it is possible to reuse the line function given above, due to the similarity between the two points. When the line function is evaluated at $\psi(Q)$ the output is an element of \mathbb{F}_{p^4} such that $((a + b\sqrt{\beta}) + (c + d\sqrt{\beta}) \sqrt[4]{\beta})$. The evaluation of the second point $(\bar{x}, -y)$ at the line function can be obtained with the transformation $((a - 2y + b\sqrt{\beta}) - (c + d\sqrt{\beta}) \sqrt[4]{\beta})$. Therefore, the calculation of the second function is effectively for free, as it simply involves two subtractions and a conjugation using the function generated by the first point.

Both functions must be multiplied by the accumulating variable $f \in \mathbb{F}_{p^4}$. It is possible to exploit the fact that the two functions are similar in form, by writing a special multiplication routine. As seen previously, a general multiplication in \mathbb{F}_{p^4} takes $9M$ using the Karatsuba technique, where M is a multiplication in \mathbb{F}_p . Let the first function f_{c_1} be equal to $f_{c_1} = (a + b\sqrt[4]{\beta})$ and the second function f_{c_2} be equal to $f_{c_2} = (c - b\sqrt[4]{\beta})$, where $a, b, c \in \mathbb{F}_{p^2}$. Then, the multiplication of f_{c_1} and f_{c_2} can be unrolled as

$$(a + b\sqrt[4]{\beta})(c - b\sqrt[4]{\beta}) = ac - b^2\sqrt{\beta} + b(c - a)\sqrt[4]{\beta}$$

Note that $(c - a) \in \mathbb{F}_p$, rather than \mathbb{F}_{p^2} . The form of the ac multiplication can also be

exploited by computing

$$ac = (e + f\sqrt{\beta})(g + f\sqrt{\beta}) = cg - 2f^2 + f(e + g)\sqrt{\beta}$$

The total cost to compute the multiplication of the line functions is $2M + S$ for the ac multiplication, as well as $2M + 2M$ for the overall multiplication, which results in $6M + S$ instead of the general cost of $9M$. When this technique is implemented, we find that although the denominator elimination method is theoretically slightly faster, the performance of denominator elimination and Algorithm 12 is roughly the same, for the genus 2 case under consideration. However, we suggest that Algorithm 12 is a more natural algorithm to use in practice, as it does not require the construction of customised multiplication routines, such as those given in this section.

6.4.3 Theoretical analysis

In this section, the theoretical cost of computing the Tate pairing using the genus 2 curve $C: y^2 = x^5 + 1$ is analysed. Firstly, the analysis of Choe and Lee [17] is reproduced. Let S, M, I be a squaring, multiplication and inversion respectively in \mathbb{F}_p . Choe and Lee estimate the cost of computing the Tate pairing (without including the cost of the final exponentiation) as

$$\log_2(n)(T_D + T_c + T_d + 2T_{sk} + 2T_{mk}) + (1/2)\log_2(n)(T_A + T_c + T_d + 2T_{mk}),$$

where

- 1 $T_D = I + 23M + 5S$ - the cost of doubling a general divisor
- 2 $T_A = I + 23M + 2S$ - the cost of adding two general divisors
- 3 $T_c + T_d = 22M + 5S$ - the cost of evaluating the line functions c and d , with a precomputation of $8M + 3S$
- 4 $T_{sk} = 8M$ - the cost of squaring in \mathbb{F}_{p^k} (where $k = 4$)

5 $T_{mk} = 9M$ - the cost of multiplication in \mathbb{F}_{p^k} (where $k = 4$)

As Choie and Lee use a random subgroup order n to compute the Tate pairing, $(1/2) \log_2(n)$ additions must be performed in Miller's algorithm. Let $\log_2(n) \approx 160$. Then evaluating the formula given above yields a total cost for computing the Tate pairing as $240I + 17688M + 2163S$.

The theoretical cost of computing the Tate pairing is now examined using the optimizations given in this chapter. Algorithm 12 is used in combination with a subgroup order of very low Hamming weight. The efficient formulae for doubling an element of $\text{Pic}_C^0(\mathbb{F}_p)$ as given in Table 6.3 are used, as well as the efficient means of constructing \mathbb{F}_{p^4} . Finally, the second argument to Miller's algorithm is defined to be a degenerate divisor, and the efficient formulae that have been derived to speed up the evaluation of this divisor at the line functions are used. Therefore, the theoretical cost for computing the Tate pairing is now given as (again without including the cost of the final exponentiation)

$$\log_2(n)(T_D + T_c + T_d + T_{s,k} + 2T_{mk}) + 2(T_A + T_c + T_d + 2T_{mk}),$$

where

1 $T_D = I + 22M + 4S$ - the cost of doubling a general divisor

2 $T_A = I + 23M + 2S$ - the cost of adding two general divisors

3 $T_c + T_d = 18M$ - the cost of evaluating the line functions c and d , with a precomputation of $17M + 1S$

4 $T_{sk} = 6M$ - the cost of squaring in \mathbb{F}_{p^k} (where $k = 4$)

5 $T_{mk} = 9M$ - the cost of multiplication in \mathbb{F}_{p^k} (where $k = 4$)

Again, let $\log_2(n) \approx 160$. Then the theoretical cost of computing the Tate pairing is given as $162I + 10375M + 645S$. This is a substantial improvement over the results of Choie and Lee. The largest single factor in this improvement is the use of a prime

Table 6.6 Theoretical complexity of Miller's algorithm

Case	Subgroup order	Complexity
1	Random [17]	240I, 17688M, 2163S
2	Solinas prime (our work)	162I, 10375M, 645S
3	NAF (our work)	214I, 13404M, 748S

subgroup order n with a very low Hamming weight. It is appropriate to examine the cost of the new formula for a random prime order n as well, to guard against a future attack that might exploit the Hamming weight of n in some manner. A random subgroup order n has a Hamming weight of $(1/2) \log_2(n)$ on average, meaning that $(1/2) \log_2(n)$ additions must take place in Miller's algorithm. This is the approach taken by Choe and Lee [17]. However, it is possible to improve on this.

Computing the opposite of an element in $\text{Pic}_C^0(\mathbb{F}_p)$ is essentially for free, as detailed in Chapter 2. Therefore, both the addition and subtraction of divisors in $\text{Pic}_C^0(\mathbb{F}_p)$ have the same computational cost. Whenever a group has this property, it is possible to exploit the Non-Adjacent Form (NAF) [91] of the subgroup order n . Let $l = \log_2(n)$. Then the (binary) NAF of n is an expansion $n = \sum_{i=0}^{l-1} n_i 2^i$, where $n_i \in \{0, \pm 1\}$, and $n_i n_{i+1} = 0$ for all $i \geq 0$. The number of non-zero terms in the Hamming weight of the NAF of n is (on average) $(1/3) \log_2(n)$, which implies a sixth less additions in Miller's algorithm than using the standard method. Therefore, combining the NAF of n with all of the optimisations introduced in this chapter gives a cost of 214I + 13404M + 748S for computing the Tate pairing. The theoretical results are summarised in Table 6.6.

6.5 Experimental Results

In this section, experimental results are given for computing the Tate pairing using the techniques detailed in this chapter for the supersingular genus 2 curve defined over \mathbb{F}_p . Three levels of security were defined for implementation, namely (160/1024), (192/2048) and (224/4096). It has been shown how to select a prime subgroup order n of the required

Table 6 7 Security Parameters

160/1024 security level
$n = 2^{159} + 2^{17} + 1$
$p = 63324531451181148200275171731203125718855624493339065310878459$ 331886717065893
192/2048 security level
$n = 2^{191} + 2^2 + 1$
$p = 89284651228083788426899503684145515482879124715345625109737480$ 602016411174689533635990672440279080762322569446999588756146485641 92943960634648749730387013
224/4096 security level
$n = 2^{223} + 2^{13} + 1$
$p = 15572288413151584018732355885170470078314521100905501866179797$ 721305996406660922169152480135059877975286648042107836950744921979 175468464339740485127309529376149370584312783605245791516787233435 196077050664154130594222494359548777260251667610641320053258135302 4750990143717859982402535061826066311255496083453

number of bits, where n is a Solinas prime with a Hamming weight of 3. A method was also detailed to select a suitable large prime p such that $p^2 + 1 \equiv 0 \pmod n$. The prime p must be congruent to $5 \pmod 8$ in order to use the finite field constructions given for \mathbb{F}_{p^2} and \mathbb{F}_{p^4} detailed earlier. Also, p must be congruent to $2, 3 \pmod 5$, as these are the conditions associated with the curve itself. Suitable values for p for the three security levels are given in Table 6 7.

Table 6 8 details the experimental results for the implementation of the Tate pairing using the (160/1024) security level. Table 6 9 gives the experimental results for the (192/2048) security level, and Table 6 10 details the timings for the (224/4096) security level. All of the timings are given in milliseconds. There are four cases in each table, all of which have a number of optimisations in common that have been derived in this chapter. These include the efficient finite field construction for \mathbb{F}_{p^4} , the explicit formulae for doubling a divisor as given in Table 6 3, and the formula given for evaluating the line function at the image divisor at each iteration of Miller's algorithm.

The first three cases in each table use the new variant of Miller's algorithm that is given

Table 6 8 Experimental results - (160/1024) security level

Case	Description	Running time (ms)
1	Evaluating at degenerate divisor	16
2	Evaluating at general divisor	20 7
3	Evaluating using Mumford rep	20 45
4	Elliptic curve timing [105]	8 9

Table 6 9 Experimental results - (192/2048) security level

Case	Description	Running time (ms)
1	Evaluating at degenerate divisor	49
2	Evaluating at general divisor	62
3	Evaluating using Mumford rep	61
4	Elliptic curve timing [105]	20 5

Table 6 10 Experimental results - (224/4096) security level

Case	Description	Running time (ms)
1	Evaluating at degenerate divisor	183
2	Evaluating at general divisor	232
3	Evaluating using Mumford rep	229
4	Elliptic curve timing [105]	85

in Algorithm 12. The first case in each table is the time taken to compute the Tate pairing when a degenerate divisor is used as the second input to Miller’s algorithm. The second case in each table gives the time for when a general divisor is used as the second input to the algorithm. The two finite points in the support of the divisor are extracted, and the line function that is generated at each iteration of the algorithm is evaluated separately at both points. The third case also computes a general pairing, except that the second input to Miller’s algorithm is in Mumford representation. This case is always faster when the finite points in the image divisor are defined over a larger field.

The fourth case in each table are timings that are given by Scott [105] using elliptic curves, and an equivalent level of security to the genus 2 timings presented here. In Table 6.8, the elliptic curve in question has an embedding degree of $k = 2$, and $\log_2(p) \approx 512$ as a result. In Table 6.9, the elliptic curve has an embedding degree of $k = 4$ and $\log_2(p) \approx 512$. In Table 6.10, the elliptic curve has an embedding degree of $k = 8$ and $\log_2(p) \approx 512$. It could be argued that comparing the (ordinary) elliptic curve case with $k = 8$ to the genus 2 case is ‘unfair’, as the theory on constructing genus 2 curves with a higher embedding degree over \mathbb{F}_p is as yet undeveloped.

A number of conclusions can be drawn from these tables. Firstly, previous experimental results are due to Choie and Lee [17], who give timings to compute the Tate pairing on this curve that range between 500 and 600 ms on a Pentium IV 2 GHz, for the (160/1024) security level. Our timings far outperform this, as demonstrated in Table 6.8. Secondly, the results given in this section indicate that genus 2 pairings over large prime fields are valid candidates for practical implementation. However, the elliptic curve timings are approximately twice as fast as the genus 2 timings for all three security levels. This is roughly what one would expect, due to the more complicated group law in the genus 2 case.

All of the experiments were performed on our platform of a Pentium IV, which has a clock speed of 2.8 GHz, and which runs version 2.6.12 of the Linux kernel. The code is written in C/C++ and is compiled using version 4.01 of the GCC/G++ compiler suite. The efficient implementation of the finite field \mathbb{F}_p is taken from MIRACL 4.85. In particular,

MIRACL supports special assembly language routines that can be used when working with prime moduli of a fixed number of bits

6.6 Conclusion

In this chapter, it has been shown that pairing calculation on supersingular genus 2 curves is efficient, and that these curves are a viable candidate for the practical implementation of pairing based cryptosystems as a result. Efficient formulae have been derived for doubling a divisor and for extracting the functions that are required in Miller's algorithm. It has been shown how to choose an optimal subgroup order with a low Hamming weight, and how to implement the finite field arithmetic efficiently. It has also been shown how the distortion map can be used to speed up the evaluation of the image divisor at the line function.

A new variant of Miller's algorithm has been introduced for hyperelliptic curves with an even embedding degree. This algorithm shows that it is never necessary to perform inversion when calculating the line functions in Miller's algorithm, even if the image divisor is not of a special form. This algorithm is interesting in two ways. First of all, it provides a nice historical bridge between the optimisations introduced by Galbraith et al [31], and those introduced by Barreto et al [5], as summarised in Table 6.5. Secondly, although this algorithm is not as fast as using denominator elimination in the general case, it can be faster when working with hyperelliptic curves of genus $g > 1$ and degenerate divisors.

A theoretical analysis of the cost of computing the Tate pairing using our optimisations has been performed, and compared to previous results in the literature. Finally, experimental results have been provided on the implementation of the Tate pairing. In particular, our timings are the fastest reported in the literature to date by a considerable margin. However, the timings show that pairing implementation on genus 2 curves over \mathbb{F}_p is about twice as slow as pairing implementation on elliptic curves over \mathbb{F}_p with an equivalent level of security. If this performance gap is to be bridged, it will be necessary to derive ordinary genus 2 curves over \mathbb{F}_p with a higher embedding degree than that offered by supersingular curves.

Chapter 7

Conclusion

7.1 Review

It was shown that the Tate pairing can be computed in an efficient manner using supersingular genus 2 curves over finite fields of characteristic 2. The best choice of curve to use was investigated, and an octupling automorphism was obtained on the selected curves. Rather than compute the functions that are required in Miller's algorithm from the Cantor composition and reduction of divisors, explicit formulae were provided that were derived using the octupling automorphism. The idea of using degenerate divisors was explored. It was shown how precomputation can be deployed to reduce the amount of computation to be performed in the algorithm itself. The Frobenius endomorphism was also exploited to calculate some of the functions required in the algorithm.

Furthermore, it was shown how it is possible to achieve a more efficient pairing calculation by utilising the η pairing construct. In the genus 2 case, the η pairing requires a longer loop size in Miller's algorithm than the standard Tate pairing. However, the η pairing has many advantages over the Tate pairing, such as the removal of additions from the loop and a final exponentiation that can be easily computed. The genus 2 η pairing is also far simpler to implement than the version of the algorithm that used the Frobenius endomorphism to expedite the computation. It was then shown how a specific instance of the η pairing can

be computed without the final exponentiation, assuming that the evaluation of the vertical line functions is included in the algorithm. This was the first time that any method for computing the Tate pairing was shown to have this property.

The truncated version of the η pairing, the η_T pairing, was then investigated. This approach halves the number of loop iterations required in Miller's algorithm compared to the η pairing. The disadvantages of the η_T pairing are that an addition must be performed at the end of the loop, and that it has a more expensive final exponentiation than the η pairing. However, techniques were described to reduce the computational impact of both of these properties. A comprehensive series of tests was then conducted, comparing the implementation of the genus 2 Tate, η and η_T pairings using different security levels. These results were compared to the efficiency of implementing the Tate pairing using supersingular elliptic curves of an equivalent level of security. The conclusion was that the genus 2 η_T pairing yields the fastest pairing implementation over finite fields of low characteristic that has been reported in the literature to date.

The implementation of the Tate pairing using a supersingular genus 2 curve over a large prime field was then described. A new variant of Miller's algorithm was derived that is more generic than the standard denominator elimination technique, and that can be useful in certain circumstances when using hyperelliptic curves of genus $g > 1$. Existing formulae for computing both the group law and the functions required for Miller's algorithm were modified and improved. It was also shown how the form of the distortion map can be exploited to evaluate the intermediate functions in Miller's algorithm more efficiently. A theoretical analysis was performed against previous work, and a wide range of timings was reported using various standard levels of security. These results were compared to existing results in the literature on pairing implementation using supersingular elliptic curves. The conclusion was that pairing implementation on genus 2 curves in this context is slower than for elliptic curves, but still competitive.

In summation, in this thesis it was demonstrated that pairing calculation using supersingular genus 2 curves can be achieved efficiently. This result implies that genus 2 curves are

a valid alternative to using elliptic curves for implementing cryptographic protocols based on pairings. This is a useful result, as it is good practice to have an alternative means to achieve anything in cryptography. In a more practical sense, this result allows protocol designers to consider a wider range of curves for pairing implementation. The actual selection of curve parameters rests on a wide range of practical considerations, such as the computing platform or the language being used. It is to be expected that elliptic curves will prove more useful for most circumstances. This is due mainly to their simple description, which allows for an easier implementation by the non-specialist. However, we believe it is likely that genus 2 pairings will be deployed in certain niches, such as embedded hardware or low-powered devices.

It is not necessary in mathematics to have any end goal in sight when considering an area in which to research. Even if one does not accept the theoretical or practical reasons given in the previous paragraph for considering genus 2 pairings, the study of genus 2 curves cannot help but improve our knowledge about pairing implementation on elliptic curves. An example from this thesis is that the inspiration for proving that there is no need for a final exponentiation for the η pairing on supersingular elliptic curves came from experimentation with the genus 2 η pairing. Much work remains to be done on both the theory and implementation of pairings on hyperelliptic curves before one can be truly confident of the security of pairing based cryptography.

7.2 Open Questions

There are a large number of open questions relating to (hyper)elliptic curve cryptography and the implementation of bilinear pairings. However, in this section only questions that arise from the work in this thesis are examined. Rubin and Silverberg [96] give an upper bound of $k = 6$ on the embedding degree of supersingular genus 2 curves over a large prime field \mathbb{F}_p . Very recently, Galbraith et al [36] derived a suitable supersingular genus 2 curve with this maximum embedding degree. However, the curve in question is a real quadratic

genus 2 curve. As pairing implementation has not as yet been attempted on such a curve, it is still an open question to derive a supersingular imaginary quadratic genus 2 curve with an embedding degree of $k = 6$ that is suitable for pairing implementation. A curve with this maximum embedding degree would give additional advantages to genus 2 pairings over \mathbb{F}_p .

Another open question is to find an ordinary genus 2 curve with a low embedding degree. It is important to do this for a number of reasons. Firstly, to parallel the current research that is being carried out on elliptic curves. The literature on ordinary elliptic curves with a low embedding degree has expanded significantly in recent years. This work has yielded a wide range of suitable curves over \mathbb{F}_p . However, there are as yet no known ordinary elliptic curves over \mathbb{F}_{2^m} with a low embedding degree (another open question in itself). Secondly, cryptographers have long had misgivings about using supersingular curves, due to a suspicion that the extra structure associated with such curves could be used in a destructive sense. For example, supersingular curves are no longer used for cryptosystems based on the DLP in $\text{Pic}_C^0(\mathbb{F}_q)$, due to the MOV/FR attack. Ordinary curves with a low embedding degree are also vulnerable to these attacks. However, it is possible that ordinary curves might be resistant to future attacks on supersingular curves in the context of pairings.

Thirdly, due to the work of Galbraith [32] and Rubin and Silverberg [96], it is known that there is a bound on the embedding degree of all supersingular hyperelliptic curves that are interesting for cryptography. This is particularly problematic over large prime fields, as only small embedding degrees can be obtained. In contrast, ordinary elliptic curves are known to exist over \mathbb{F}_p with a large range of embedding degrees that are useful for implementation, such as $k = 12$ and $k = 24$. This is a significant advantage associated with ordinary curves, and it would be extremely useful to replicate this work in the genus 2 context. Another reason to consider ordinary curves is the paucity of suitable supersingular curves for pairing based cryptography over a given finite field. It is desirable to be able to generate curves in a provably random fashion in order to generate confidence that the curve equation is not weak in some way.

Fourthly, the recent work of Hess et al [48] in deriving the Ate pairing shows that

pairing implementation on ordinary elliptic curves could be even more efficient than using supersingular elliptic curves. It is to be assumed that this result holds for ordinary genus 2 curves, however in the absence of any suitable curves it remains an open question. The recent theory of the Ate pairing is the most powerful and comprehensive theoretical examination of the computation of the Tate pairing in recent years. However, there has been little extension or examination of the Ate pairing in the literature, and it is another open question as to whether the Ate pairing can be developed further. Pairing implementation using both elliptic and genus 2 curves is fast approaching scalar multiplication in terms of efficiency. It is possible that some modification of the Ate pairing might even enable pairing implementation to become faster than a general scalar multiplication.

In chapter 5 of this thesis, it was shown that the final exponentiation required to compute the η pairing can be avoided for certain curves, as long as the vertical line functions are included. However, the mathematical proofs that are provided are unsatisfactory, as they do not address the more general question as to why this property holds. A proof that all η pairings are bilinear without the final exponentiation would be an interesting result. A more specific and pressing question is to prove the genus 2 η case, as we have been unable to achieve this as yet. An even more interesting question is whether this idea can be applied to the Ate pairing using ordinary elliptic curves over \mathbb{F}_p . As the final exponentiation is generally expensive over \mathbb{F}_p , eliminating it using our technique might be more efficient than using denominator elimination.

It may also be worth reconsidering the value of using hyperelliptic curves of higher genera for pairings. In particular, little work has been done on using hyperelliptic curves of genus 3. Gaudry et al [40] recommend increasing the group size of hyperelliptic genus 3 curves by 12.5% to take their index-calculus attack into account. However, this is not necessarily an impediment to using these curves for pairing based cryptography, if it can be shown that pairing calculation on hyperelliptic curves of genus 3 can be achieved in an efficient manner. In particular, the Ate pairing might yield an efficient pairing implementation on genus 3 hyperelliptic curves.

Bibliography

- [1] R. M. Avanzi. Aspects of Hyperelliptic Curves over Large Prime Fields in Software Implementations. In *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 148–162. Springer-Verlag, 2004.
- [2] R. M. Avanzi. The Complexity of Certain Multi-Exponentiation Techniques in Cryptography. *Journal of Cryptology*, 18(4): 357–373, 2005.
- [3] P. S. L. M. Barreto. A Note on Efficient Computation of Cube Roots in Characteristic 3. *Cryptology ePrint Archive*, Report 2004/305, 2004. Available from <http://eprint.iacr.org/2004/305>.
- [4] P. S. L. M. Barreto, S. D. Galbraith, C. O. hEigeartaigh, and M. Scott. Efficient Pairing Computation on Supersingular Abelian Varieties. *Cryptology ePrint Archive*, Report 2004/375, 2004. Available from <http://eprint.iacr.org/2004/375>.
- [5] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient Algorithms for Pairing-Based Cryptosystems. In *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer-Verlag, 2002.
- [6] P. S. L. M. Barreto, B. Lynn, and M. Scott. On the Selection of Pairing-Friendly Groups. In *Selected Areas in Cryptography – SAC 2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 17–25. Springer-Verlag, 2003.

- [7] P S L M Barreto, B Lynn, and M Scott Efficient Implementation of Pairing-Based Cryptosystems *Journal of Cryptology*, 17(4) 321–334, 2004
- [8] I F Blake, V K Murty, and G Xu Refinement of Miller’s algorithm for Computing the Weil/Tate Pairing *Journal of Algorithms*, 58(2) 134–149, 2006
- [9] I F Blake, G Seroussi, and N P Smart *Advances in Elliptic Curve Cryptography* Cambridge University Press, 2005
- [10] B den Boer Diffie-Hellman is as Strong as Discrete Log for Certain Primes In *Advances in Cryptology – CRYPTO 1988*, volume 403 of *Lecture Notes in Computer Science*, pages 530–539 Springer-Verlag, 1990
- [11] D Boneh and M Franklin Identity-Based Encryption from the Weil Pairing *SIAM Journal of Computing*, 32(3) 586–615, 2003
- [12] D Boneh, B Lynn, and H Shacham Short Signatures from the Weil Pairing In *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532 Springer-Verlag, 2001
- [13] D Boneh and A Silverberg Applications of Multilinear forms to Cryptography *Contemporary Mathematics*, 324 71–90, 2003
- [14] D G Cantor Computing in the Jacobian of a Hyperelliptic Curve *Mathematics of Computation*, 48(177) 95–101, 1987
- [15] S Chatterjee, P Sarkar, and R Barua Efficient Computation of Tate Pairing in Projective Coordinate over General Characteristic Fields In *Information Security and Cryptology – ICISC 2004*, volume 3506 of *Lecture Notes in Computer Science*, pages 168–181 Springer-Verlag, 2005
- [16] Y Choie, E Jeong, and E Lee Supersingular Hyperelliptic Curves of Genus 2 over Finite Fields *Journal of Applied Mathematics and Computation*, 163(2) 565–576, 2005

- [17] Y Choie and E Lee Implementation of Tate Pairing on Hyperelliptic Curves of Genus 2 In *Information Security and Cryptology – ICISC 2003*, volume 2971 of *Lecture Notes in Computer Science*, pages 97–111 Springer-Verlag, 2004
- [18] Y Choie and D Yun Isomorphism Classes of Hyperelliptic Curves of Genus 2 over \mathbb{F}_q In *Australasian Conference on Information Security and Privacy – ACISP 2002*, volume 2384 of *Lecture Notes in Computer Science*, pages 190–202 Springer-Verlag, 2002
- [19] H Cohen and G Frey, editors *Handbook of Elliptic and Hyperelliptic Curve Cryptography* Chapman & Hall/CRC, 2006
- [20] S A Cook On the Minimum Computation Time of Functions Ph D Thesis, Harvard University Department of Mathematics, 1966
- [21] D Coppersmith Fast Evaluation of Logarithms in Fields of Characteristic Two *IEEE Transactions on Information Theory*, 30(4) 587–594, 1984
- [22] W Diffie and M Hellman New Directions in Cryptography *IEEE Transactions on Information Theory*, 22 644–654, 1976
- [23] I Duursma and H-S Lee Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$ In *Advances in Cryptology – ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 111–123 Springer-Verlag, 2003
- [24] K Eisentrager, K Lauter, and P L Montgomery Fast Elliptic Curve Arithmetic and Improved Weil Pairing Evaluation In *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 343–354 Springer-Verlag, 2003
- [25] K Eisentrager, K Lauter, and P L Montgomery Improved Weil and Tate Pairings for Elliptic and Hyperelliptic Curves In *Algorithmic Number Theory Symposium – ANTS VI*, volume 3076 of *Lecture Notes in Computer Science*, pages 169–183 Springer-Verlag, 2004

- [26] T ElGamal A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms *IEEE Transactions on Information Theory*, 31(4) 469–472, 1985
- [27] K Fong, D Hankerson, J Lopez, and A Menezes Field Inversion and Point Halving Revisited *IEEE Transactions on Computers*, 53(8) 1047–1059, 2004
- [28] D Freeman Fast Arithmetic and Pairing Evaluation on Genus 2 Curves Preprint, 2005 Available from <http://math.berkeley.edu/~dfreeman/papers/genus2.pdf>
- [29] G Frey and T Lange Fast Bilinear Maps from the Tate-Lichtenbaum Pairing on Hyperelliptic Curves In *Algorithmic Number Theory Symposium – ANTS VII*, volume 4076 of *Lecture Notes in Computer Science*, pages 466–479 Springer-Verlag, 2006
- [30] G Frey and H-G Ruck A Remark Concerning m -Divisibility and the Discrete Logarithm Problem in the Divisor Class Group of Curves *Mathematics of Computation*, 62(206) 865–874, 1994
- [31] S Galbraith, K Harrison, and D Soldera Implementing the Tate Pairing In *Algorithmic Number Theory Symposium – ANTS V*, volume 2369 of *Lecture Notes in Computer Science*, pages 324–337 Springer-Verlag, 2002
- [32] S D Galbraith Supersingular Curves in Cryptography In *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 495–513 Springer-Verlag, 2001
- [33] S D Galbraith, C O hEigearthaigh, and C Sheedy Simplified Pairing Computation and Security Implications *Cryptology ePrint Archive*, Report 2006/169, 2006 <http://eprint.iacr.org/2006/169>
- [34] S D Galbraith and A Menezes Algebraic Curves and Cryptography *Finite Fields and applications*, 11(3) 544–577, 2005

- [35] S D Galbraith, K G Paterson, and N P Smart Pairings for Cryptographers Cryptology ePrint Archive, Report 2006/165, 2006 <http://eprint.iacr.org/2006/165>
- [36] S D Galbraith, J Pujolas, C Ritzenthaler, and B Smith Distortion Maps for Genus Two Curves Cryptology ePrint Archive, Report 2006/375, 2006 <http://eprint.iacr.org/2006/375>
- [37] R P Gallant, R J Lambert, and S A Vanstone Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200 Springer-Verlag, 2001
- [38] P Gaudry Fast Genus 2 Arithmetic Based on Theta Functions Cryptology ePrint Archive, Report 2005/314, 2005 <http://eprint.iacr.org/2005/314>
- [39] P Gaudry, F Hess, and N P Smart Constructive and Destructive Facets of Weil Descent on Elliptic Curves *Journal of Cryptology*, 15(1) 19–46, 2002
- [40] P Gaudry, E Thome, N Thériault, and C Diem A Double Large Prime Variation for Small Genus Hyperelliptic Index Calculus *Mathematics of Computation*, 76(257) 475–492, 2007
- [41] G van der Geer and M van der Vlugt Reed-Muller Codes and Supersingular Curves I *Compositio Mathematica*, 84(3) 333–367, 1992
- [42] R Granger, D Page, and N P Smart High Security Pairing-Based Cryptography Revisited In *Algorithmic Number Theory Symposium – ANTS VII*, volume 4076 of *Lecture Notes in Computer Science*, pages 480–494 Springer-Verlag, 2006
- [43] R Granger, D Page, and M Stam On Small Characteristic Algebraic Tori in Pairing-Based Cryptography *LMS Journal of Computation and Mathematics*, 9 64–85, 2006

- [44] C Gunther, T Lange, and A Stein Speeding up the Arithmetic on Koblitz Curves of Genus Two In *Selected Areas in Cryptography – SAC 2000*, volume 2012 of *Lecture Notes in Computer Science*, pages 106–117 Springer-Verlag, 2001
- [45] R Harasawa, Y Sueyoshi, and A Kudo Tate Pairing for $y^2 = x^5 - \alpha x$ in Characteristic Five In *Symposium on Cryptography and Information Security – SCIS 2005*, pages 931–935, 2005
- [46] C O hEigeartaigh Speeding up Pairing Computation Cryptology ePrint Archive, Report 2005/293, 2005 [http //eprint iacr org/2005/293](http://eprint.iacr.org/2005/293)
- [47] C O hEigeartaigh and M Scott Pairing Calculation on Supersingular Genus 2 Curves Cryptology ePrint Archive, Report 2006/005, 2006 [http //eprint iacr org/2006/005](http://eprint.iacr.org/2006/005)
- [48] F Hess, N P Smart, and F Vercauteren The Eta Pairing Revisited *IEEE Transactions on Information Theory*, 52(10) 4595–4602, 2006
- [49] M Hietalahti Hyperelliptic Curves and their Use in Cryptosystems, a Literature Survey, 2001 Available from [http //www tcs hut fi/~mhietala/hype ps](http://www.tcs.hut.fi/~mhietala/hype.ps)
- [50] L Hu, J-W Dong, and D-Y Pei Implementation of Cryptosystems based on Tate Pairing *Journal of Computer Science and Technology*, 20(2) 264–269, 2005
- [51] T Izu and T Takagi Efficient Computations of the Tate Pairing for the Large MOV Degrees In *Information Security and Cryptology – ICISC 2002*, volume 2587 of *Lecture Notes in Computer Science*, pages 283–297 Springer-Verlag, 2003
- [52] A Joux A One-Round Protocol for Tripartite Diffie-Hellman In *Algorithmic Number Theory Symposium – ANTS IV*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394 Springer-Verlag, 2000

- [53] A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Computational Diffie-Hellman in Cryptographic Groups. *Journal of Cryptology*, 16(4):239–247, 2003.
- [54] M. Joye and J. J. Quisquater. Efficient Computation of Full Lucas Sequences. *Electronics Letters*, 32(6):537–538, 1996.
- [55] M. Joye and S. Yen. The Montgomery Powering Ladder. In *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer-Verlag, 2003.
- [56] M. Jacobson Jr., A. Menezes, and A. Stein. Hyperelliptic Curves and Cryptography. In *Fields Institute Communications Series*, volume 41, pages 255–282, 2004.
- [57] B. G. Kang and J. H. Park. Powered Tate Pairing Computation. *Cryptology ePrint Archive*, Report 2005/260, 2005. <http://eprint.iacr.org/2005/260>
- [58] B. G. Kang and J. H. Park. On the Relationship between Squared Pairings and Plain Pairings. *Information Processing Letters*, 97(6):219–224, 2006.
- [59] A. A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata. *Soviet Physics Doklady*, 7:595–596, 1963.
- [60] M. Katagi, T. Akishita, I. Kitamura, and T. Takagi. Some Improved Algorithms for Hyperelliptic Curve Cryptosystems using Degenerate Divisors. In *Information Security and Cryptology – ICISC 2004*, volume 3506 of *Lecture Notes in Computer Science*, pages 296–312. Springer-Verlag, 2005.
- [61] M. Katagi, I. Kitamura, T. Akishita, and T. Takagi. Novel Efficient Implementations of Hyperelliptic Curve Cryptosystems using Degenerate Divisors. In *Information Security Applications – WISA 2004*, volume 3325 of *Lecture Notes in Computer Science*, pages 345–359. Springer-Verlag, 2004.

- [62] F Kobayashi, K Aoki, and H Imai Efficient Algorithms for Tate Pairing *IEICE Transactions Fundamentals*, E89-A(1) 134–143, January 2006
- [63] N Koblitz Elliptic Curve Cryptosystems *Mathematics of Computation*, 48 203–209, 1987
- [64] N Koblitz Hyperelliptic Cryptosystems *Journal of Cryptology*, 1(3) 139–150, 1989
- [65] N Koblitz and A Menezes Pairing-Based Cryptography at High Security Levels In *Cryptography and Coding – IMA 2005*, volume 3796 of *Lecture Notes in Computer Science*, pages 13–36 Springer-Verlag, 2005
- [66] S Kwon Efficient Tate Pairing Computation for Elliptic Curves over Binary Fields In *Australasian Conference on Information Security and Privacy – ACISP 2005*, volume 3574 of *Lecture Notes in Computer Science*, pages 134–145 Springer-Verlag, 2005
- [67] T Lange Fast Arithmetic on Hyperelliptic Curves Ph D Thesis, Universitat-Gesamthochschule Essen, 2002
- [68] T Lange Inversion-Free Arithmetic on Genus 2 Hyperelliptic Curves *Cryptology ePrint Archive*, Report 2002/147, 2002 <http://eprint.iacr.org/2002/147>
- [69] T Lange Weighted Coordinates on Genus 2 Hyperelliptic Curves *Cryptology ePrint Archive*, Report 2002/153, 2002 <http://eprint.iacr.org/2002/153>
- [70] T Lange Formulae for Arithmetic on Genus 2 Hyperelliptic Curves *Applicable Algebra in Engineering Communication and Computing*, 15(5) 295–328, 2005
- [71] E Lee, H-S Lee, and Y Lee Fast Computation of Tate Pairing on General Divisors of Genus 3 Hyperelliptic Curves *Cryptology ePrint Archive*, Report 2006/125, 2006 <http://eprint.iacr.org/2006/125>

- [72] E Lee and Y Lee Tate Pairing Computation on the Divisors of Hyperelliptic Curves for Cryptosystems Cryptology ePrint Archive, Report 2005/166, 2005 <http://eprint.iacr.org/2005/166>
- [73] A K Lenstra Unbelievable Security Matching AES Security using Public Key Systems In *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 67–86 Springer-Verlag, 2001
- [74] A K Lenstra and E R Verheul Selecting Cryptographic Key Sizes *Journal of Cryptology*, 14(4) 255–293, 2001
- [75] R Lidl and H Niederreiter *Finite Fields* Number 20 in *Encyclopedia of Mathematics and its Applications* Cambridge University Press, 2nd edition, 1997
- [76] U M Maurer Towards the Equivalence of Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms In *Advances in Cryptology – CRYPTO 1994*, volume 839 of *Lecture Notes in Computer Science*, pages 271–281 Springer-Verlag, 1994
- [77] U M Maurer and S Wolf The Relationship Between Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms *SIAM Journal on Computing*, 28(5) 1689–1721, 1999
- [78] A Menezes *Elliptic Curve Public Key Cryptosystems* Kluwer Academic Publishers, 1993
- [79] A Menezes An Introduction to Pairing-Based Cryptography Unpublished manuscript, 2005 <http://www.math.uwaterloo.ca/~ajmenez/publications/pairings.pdf>
- [80] A Menezes, T Okamoto, and S A Vanstone Reducing Elliptic Curve Logarithms to a Finite Field *IEEE Transactions on Information Theory*, 39(5) 1639–1646, 1993

- [81] A Menezes, Y Wu, and R Zuccherato An Elementary Introduction to Hyperelliptic Curves *Appendix in Algebraic Aspects of Cryptography by Neal Koblitz*, pages 155–178, 1998
- [82] V S Miller Short Programs for Functions on Curves Unpublished manuscript, 1986 <http://crypto.stanford.edu/miller/miller.pdf>
- [83] V S Miller Use of Elliptic Curves in Cryptography In *Advances in Cryptology – CRYPTO 1985*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426 Springer-Verlag, 1986
- [84] V S Miller The Weil Pairing and its Efficient Calculation *Journal of Cryptology*, 17(4) 235–261, 2004
- [85] Y Miyamoto, H Doi, K Matsuo, J Chao, and S Tsuji A Fast Addition Algorithm of Genus Two Hyperelliptic Curve In *Symposium on Cryptography and Information Security – SCIS 2002*, pages 497–502, 2002 (in Japanese)
- [86] D Mumford *Tata lectures on Theta II* Birkhauser, 1984
- [87] V Nechaev Complexity of a Determinate Algorithm for the Discrete Logarithm *Mathematical Notes*, 55(2) 165–172, 1994
- [88] C M Park, M H Kim, and M Yung A Remark on Implementing the Weil Pairing In *CISC 2005*, volume 3822 of *Lecture Notes in Computer Science*, pages 313–323 Springer-Verlag, 2005
- [89] S Pohlig and M Hellman An Improved Algorithm for Computing Logarithms over $GF(p)$ and its Cryptographic Significance *IEEE Transactions on Information Theory*, 24(1) 106–110, 1978
- [90] J Pollard Monte Carlo Methods for Index Computation mod p *Mathematics of Computation*, 32 918–924, 1978

- [91] G W Reitwiesner Binary Arithmetic *Advances in Computers*, 1 231–308, 1960
- [92] P Ribenboim *Classical Theory of Algebraic Numbers* Springer-Verlag, 2001
- [93] R Rivest, A Shamir, and L Adleman A Method for Obtaining Digital Signatures and Public-Key Cryptosystems *Communications of the ACM*, 21(2) 120–126, 1978
- [94] R Ronan, C O hEigeartaigh, C Murphy, M Scott, and T Kerins A Hardware Accelerator for a Pairing-Based Cryptosystem To appear in a special issue of the *Journal of Systems Architecture*
- [95] R Ronan, C O hEigeartaigh, C Murphy, M Scott, T Kerins, and W P Marnane An Embedded Processor for a Pairing-Based Cryptosystem In *Information Technology New Generations – ITNG 2006*, pages 192–197 IEEE Computer Society, 2006
- [96] K Rubin and A Silverberg Supersingular Abelian Varieties in Cryptology In *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 336–353 Springer-Verlag, 2002
- [97] K Rubin and A Silverberg Torus-Based Cryptography In *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 349–365 Springer-Verlag, 2003
- [98] H-G Ruck On the Discrete Logarithm in the Divisor Class Group of Curves *Mathematics of Computation*, 68(226) 805–806, 1999
- [99] R Sakai, K Ohgishi, and M Kasahara Cryptosystems based on Pairings In *Proceedings of the 2000 Symposium on Cryptography and Information Security Okinawa Japan*, pages 26–28, 2000
- [100] C Schnorr Efficient Signature Generation by Smart Cards *Journal of Cryptology*, 4(3) 161–174, 1991
- [101] R Schoof Elliptic Curves over Finite Fields and the Computation of Square Roots mod p *Mathematics of Computation*, 44 483–494, 1985

- [102] M. Scott. MIRACL (Multiprecision Integer and Rational Arithmetic C/C++ Library). Available from <http://indigo.ie/~mscott/>
- [103] M. Scott. Computing the Tate Pairing. In *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer-Verlag, 2005.
- [104] M. Scott. Faster pairings using an Elliptic Curve with an Efficient Endomorphism. In *Progress in Cryptology – INDOCRYPT 2005*, volume 3797 of *Lecture Notes in Computer Science*, pages 258–269. Springer-Verlag, 2005.
- [105] M. Scott. Scaling Security in Pairing-Based Protocols. Cryptology ePrint Archive, Report 2005/139, 2005. <http://eprint.iacr.org/2005/139>
- [106] M. Scott and P. Barreto. Compressed Pairings. In *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 140–156. Springer-Verlag, 2004.
- [107] A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Advances in Cryptology – CRYPTO 1984*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer-Verlag, 1985.
- [108] V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In *Advances in Cryptology – EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer-Verlag, 1997.
- [109] J. Solinas. Generalized Mersenne Numbers. Technical Report CORR 99-39, University of Waterloo, 1999. Available from <http://www.cacr.math.uwaterloo.ca/techreports/1999/corr99-39.pdf>
- [110] J. Solinas. ID-based Digital Signature Algorithms, 2003. Available from <http://www.cacr.math.uwaterloo.ca/conferences/2003/ecc2003/solinas.pdf>

- [111] M Takahashi Improving Harley Algorithms for Jacobians of Genus 2 Hyperelliptic Curves In *Symposium on Cryptography and Information Security – SCIS 2002*, 2002 (in Japanese)
- [112] A L Toom The Complexity of a Scheme of Functional Elements realizing the Multiplication of Integers *Soviet Mathematics*, 4(3) 714–716, 1963
- [113] E Verheul Evidence that XTR is More Secure than Supersingular Elliptic Curve Cryptosystems In *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 195–210 Springer-Verlag, 2001

Appendix A

Formulae

A.1 Absorbing powers of 8 for the genus 2 η pairing

In Chapter 5, the genus 2 η pairing is given as

$$\eta(P, Q) = \prod_{i=0}^{m-1} f_{8^i P}(\psi(Q))^{8^{m-1-i}},$$

where $f_{8^i P} = \alpha\beta$. In Chapter 4, it was shown how it is possible to precompute all of the powers of x_P and y_P that are required in Miller's algorithm. The goal of this section is to show how the exponentiation to 8^{m-1-i} can be brought into the formulae for α and β . This optimisation avoids the need to explicitly octuple the accumulating variable f each iteration of the loop. Therefore, rather than compute α and β each iteration, formulae are derived to compute $\alpha^{2^{3(m-1-i)}}$ and $\beta^{2^{3(m-1-i)}}$.

Calculating this efficiently requires the precomputation of certain powers of x_Q and y_Q , in addition to the precomputation of the powers of x_P and y_P mentioned above. Two arrays of size m are constructed, such that each index i in the arrays consists of the value $x_Q^{2^i}$ and $y_Q^{2^i}$. The first step in building the exponentiation into α and β is to examine how w and s_0 behave under powering by $2^{3(m-1-i)}$. Recall that $w^8 = w + 1$. As m is defined to be odd, then $m - 1 - i \equiv i \pmod{2}$, and thus $w^{2^{3(m-1-i)}} = w + \gamma_1(i)$, where $\gamma_1(i)$ is 1 when i is

odd and 0 otherwise. Also note that as $s_0^2 + s_0 = w^5 + w^3$, then $s_0^8 = s_0 + w^2$, $s_0^{8^2} = s_0 + 1$ and $s_0^{8^3} = s_0 + w^2 + 1$. This can be generalised, so that when $m \equiv 1 \pmod{4}$

$$s_0^{2^{3(m-1-i)}} = s_0 + \gamma_1(i)w^2 + \gamma_3(i),$$

and when $m \equiv 3 \pmod{4}$

$$s_0^{2^{3(m-1-i)}} = s_0 + \gamma_1(i)w^2 + \gamma_3(i) + 1,$$

where $\gamma_3(i) = 1$ when $i \equiv 1, 2 \pmod{4}$, and 0 otherwise. Let $\gamma_4(m, i)$ denote the value $\gamma_3(i)$ when $m \equiv 1 \pmod{4}$, and $\gamma_3(i) + 1$ otherwise.

As before, we write $x^{(i)} = x^{2^i}$, where i is considered modulo m . Using the basis given in Chapter 4 for elements of $\mathbb{F}_{2^{12m}}$, the constant term of $\alpha^{2^{3(m-1-i)}}$ is

$$\begin{aligned} & y_Q^{(3m-2-3i)} + \left(x_Q^{(3m-2-3i)}\right)^3 + \left(x_P^{(3i+1)} + x_P^{(3i)}\right) x_Q^{(3m-1-3i)} + \gamma_1(i) x_P^{(3i+1)} + \\ & \left(x_P^{(3i)} + \gamma_1(i) + 1\right) x_Q^{(3m-2-3i)} + y_P^{(3i)} + \gamma_3(i) + 1 \end{aligned}$$

When $m - 1 - i$ (and hence i) is odd, another term must be added to this. This term is written as

$$\gamma_1(i) \left(x_Q^{(3m-2-3i)} + 1 + \gamma_1(i) + x_P^{(3i+1)}\right) + \gamma_4(m, i)$$

This can be simplified by writing $\gamma_1(i)(1 + \gamma_1(i)) = 0$, cancelling various terms and simplifying the cubing of $x_Q^{(3m-2-3i)}$. The constant term of $\alpha^{2^{3(m-1-i)}}$ is then

$$\begin{aligned} & y_Q^{(3m-2-3i)} + \left(r_P^{(3i+1)} + r_P^{(3i)}\right) r_Q^{(3m-1-3i)} + \left(r_P^{(3i)} + 1 + r_Q^{(3m-1-3i)}\right) r_Q^{(3m-2-3i)} + \\ & y_P^{(3i)} + \gamma_5(m), \end{aligned}$$

where $\gamma_5(m) = 1$ if $m \equiv 1 \pmod{4}$ and 0 otherwise. The remaining terms are

$$\begin{aligned} & \left(x_Q^{(3m-1-3i)} + x_Q^{(3m-2-3i)} \right) w + \left(x_Q^{(3m-1-3i)} + x_P^{(3i)} + 1 \right) w^2 + \\ & \left(x_P^{(3i+1)} + x_P^{(3i)} \right) w^4 + s_0 \end{aligned}$$

The same process is now repeated for β . The constant term of $\beta^{2^{3(m-1-i)}}$ is given as

$$\begin{aligned} & y_Q^{(3m-3-3i)} + \left(x_P^{(3i+2)} + \gamma_1(i) \right) x_Q^{(3m-2-3i)} + \left(x_P^{(3i+2)} + x_P^{(3i+1)} \right) x_Q^{(3m-3-3i)} + \\ & y_P^{(3i+1)} + r_P^{(3i+1)} \left(1 + \gamma_1(i) + r_P^{(3i+2)} \right) + \gamma_3(i) + 1, \end{aligned}$$

with the addition of the term

$$\gamma_1(i) \left(x_P^{(3i+1)} + x_Q^{(3m-2-3i)} + \gamma_1(i) + 1 \right) + \gamma_4(m, i)$$

Simplifying this gives a constant term

$$\begin{aligned} & y_Q^{(3m-3-3i)} + \left(x_P^{(3i+1)} + x_Q^{(3m-2-3i)} \right) x_P^{(3i+2)} + \left(x_P^{(3i+2)} + x_P^{(3i+1)} \right) x_Q^{(3m-3-3i)} + \\ & y_P^{(3i+1)} + x_P^{(3i+1)} + \gamma_5(m) \end{aligned}$$

The remaining terms are

$$\begin{aligned} & \left(r_P^{(3i+2)} + r_P^{(3i+1)} \right) w + \left(r_Q^{(3m-3-3i)} + r_P^{(3i+2)} + 1 \right) w^2 + \\ & \left(x_Q^{(3m-2-3i)} + x_Q^{(3m-3-3i)} \right) w^4 + s_0 \end{aligned}$$

A.2 Absorbing powers of 8 for the genus 2 η_T pairing

In Chapter 5, the octupling loop of the genus 2 η_T pairing on the points P and Q is

$$\prod_{i=0}^{(m-3)/2} f_{8 \lceil 8^i \rceil P}(\psi(Q))^{2^{(3m-9-6i)/2}},$$

where $f_8^{[8^i]P} = \alpha\beta$. The goal of this section is to modify the formulae for α and β given in Chapter 4 to avoid having to octuple the accumulating variable at each iteration of Miller's algorithm. This is done by absorbing the exponentiation to $2^{3(m-3-2i)/2}$ into the formulae for α and β . As in the previous section, computing this efficiently requires the precomputation of the values $x_Q^{2^i}$ and $y_Q^{2^i}$ for all $0 \leq i \leq m-1$. Firstly, it is necessary to examine how w and s_0 behave under powering by $2^{3(m-3-2i)/2}$. As m is odd, and hence $(m-3-2i)/2 \equiv i \pmod{2}$, then $w^{2^{3(m-3-2i)/2}} = w + \gamma_1(i)$ as before. The values for $s_0^{2^{3(m-3-2i)/2}}$ are also the same as the values given for $s_0^{2^{3(m-1-i)}}$ in the previous section.

As before, we write $x^{(i)} = x^{2^i}$, where i is considered modulo m . Using the basis given in Chapter 4 for elements of $\mathbb{F}_{2^{12m}}$, the constant term of $\alpha^{2^{3(m-3-2i)/2}}$ is

$$\begin{aligned} & \left(x_Q^{((3m-7-6i)/2)} \right)^3 + \left(x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} \right) x_Q^{((3m-6i-5)/2)} + \\ & y_Q^{((3m-7-6i)/2)} + \left(x_P^{((3m-3+6i)/2)} + 1 + \gamma_1(i) \right) x_Q^{((3m-7-6i)/2)} + y_P^{((3m-3+6i)/2)} + \\ & \gamma_1(i) x_P^{((3m-1+6i)/2)} + \gamma_3(i) + 1, \end{aligned}$$

with the addition of the term

$$\gamma_1(i) \left(x_Q^{(3m-7-6i)/2} + 1 + \gamma_1(i) + x_P^{((3m-1+6i)/2)} \right) + \gamma_4(m, i)$$

Adding these two terms together and simplifying gives

$$\begin{aligned} & y_Q^{((3m-7-6i)/2)} + \left(x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} \right) x_Q^{((3m-5-6i)/2)} + y_P^{((3m-3+6i)/2)} \\ & \left(x_P^{((3m-3+6i)/2)} + 1 + x_Q^{((3m-5-6i)/2)} \right) x_Q^{((3m-7-6i)/2)} + \gamma_5(i), \end{aligned}$$

where $\gamma_5(i) \equiv 1$ if $i \equiv 1 \pmod{4}$ and 0 otherwise. The remaining terms are

$$\begin{aligned} & \left(x_Q^{((3m-5-6i)/2)} + x_Q^{((3m-7-6i)/2)} \right) w + \left(x_Q^{((3m-5-6i)/2)} + x_P^{((3m-3+6i)/2)} + 1 \right) w^2 + \\ & \left(x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} \right) w^4 + s_0 \end{aligned}$$

The exponentiation on β is now examined. The constant term of $\beta^{2^{3((m-3-2i)/2)}}$ is

$$\begin{aligned} & y_Q^{((3m-9-6i)/2)} + \left(x_P^{((3m+1+6i)/2)} + \gamma_1(i) \right) x_Q^{((3m-7-6i)/2)} + y_P^{((3m-1+6i)/2)} + \\ & \left(x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)} \right) x_Q^{((3m-9-6i)/2)} + \\ & x_P^{((3m-1+6i)/2)} \left(x_P^{((3m+1+6i)/2)} + \gamma_1(i) + 1 \right) + \gamma_3(i) + 1, \end{aligned}$$

with the addition of the term

$$\gamma_1(i) \left(x_P^{((3m-1+6i)/2)} + x_Q^{((3m-7-6i)/2)} + \gamma_1(i) + 1 \right) + \gamma_4(m, i)$$

Performing this addition and simplifying yields the constant term

$$\begin{aligned} & y_Q^{((3m-9-6i)/2)} + \left(x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)} \right) x_Q^{((3m-9-6i)/2)} + \\ & y_P^{((3m-1+6i)/2)} + x_P^{((3m+1+6i)/2)} \left(x_P^{((3m-1+6i)/2)} + x_Q^{((3m-7-6i)/2)} \right) + \\ & x_P^{((3m-1+6i)/2)} + \gamma_5(i) \end{aligned}$$

The remaining terms are

$$\begin{aligned} & \left(x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)} \right) w + \left(x_P^{((3m+1+6i)/2)} + x_Q^{((3m-9-6i)/2)} + 1 \right) w^2 + \\ & \left(x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)} \right) w^4 + s_0 \end{aligned}$$

A 3 Unrolling the $\alpha\beta$ multiplication

In this section it is shown how to multiply two special elements of $\mathbb{F}_{2^{12m}}$ in an efficient manner, by exploiting the fact that both elements have a large number of zeros as coefficients. Let $\alpha = a + bw + cw^2 + dw^4 + s_0$ and $\beta = e + fw + gw^2 + hw^4 + s_0$, where $\alpha, \beta \in \mathbb{F}_{2^{12m}}$ are written in the basis that was constructed in chapter 4. The multiplication

of α and β can then be written as

$$\begin{aligned}
\alpha\beta &= (a + bw + cw^2 + dw^4 + s_0)(c + fw + gw^2 + hw^4 + s_0) \\
&= ae + afw + agw^2 + ahw^4 + as_0 + bew + bfw^2 + bgw^3 + \\
&\quad bhw^5 + bws_0 + cew^2 + cfw^3 + cgw^4 + ch(w^5 + w^3 + w^2 + 1) + \\
&\quad cw^2s_0 + dew^4 + dfw^5 + dg(w^5 + w^3 + w^2 + 1) + dh(w + 1) + \\
&\quad dw^4s_0 + es_0 + fws_0 + gw^2s_0 + hw^4s_0 + (s_0 + w^5 + w^3)
\end{aligned}$$

Grouping all of the relevant terms together gives

$$\begin{aligned}
\alpha\beta &= (ae + ch + dg + dh) + (af + be + dh)w + (ag + bf + ce + ch + dg)w^2 + \\
&\quad (bg + cf + ch + dg + 1)w^3 + (ah + cg + de)w^4 + \\
&\quad (bh + ch + df + dg + 1)w^5 + (a + e + 1)s_0 + (b + f)ws_0 + \\
&\quad (c + g)w^2s_0 + (d + h)w^4s_0
\end{aligned}$$

This costs 16 multiplications in \mathbb{F}_{2^m} , which is a vast improvement on the 54 multiplications in \mathbb{F}_{2^m} required for a general multiplication in $\mathbb{F}_{2^{12m}}$. However, it is possible to save a further number of multiplications, by exploiting Karatsuba-like optimisations. First of all precompute the following values $dh = d \cdot h, dg = d \cdot g, ch = c \cdot h, cg = c \cdot g, ae = a \cdot e, bf = b \cdot f$. Then the multiplication is computed as

$$\begin{aligned}
\alpha\beta &= (ae + ch + dg + dh) + ((a + b)(f + e) + ae + bf + dh)w + \\
&\quad ((a + c)(g + e) + ae + cg + bf + ch + dg)w^2 + \\
&\quad ((b + c)(g + f) + bf + cg + ch + dg + 1)w^3 + \\
&\quad ((a + d)(h + e) + ae + dh + cg)w^4 + \\
&\quad ((b + d)(h + f) + bf + dh + ch + dg + 1)w^5 + \\
&\quad (a + e + 1)s_0 + (b + f)ws_0 + (c + g)w^2s_0 + (d + h)w^4s_0
\end{aligned}$$

Therefore, the total cost of the $\alpha\beta$ multiplication is only 11 multiplications in \mathbb{F}_{2^m}

t