

Generalized Mersenne Numbers in Pairing-Based Cryptography

by
Greg Zaverucha

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
July 2006

© Copyright by Greg Zaverucha, 2006

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled “ Generalized Mersenne Numbers in Pairing-Based Cryptography” by Greg Zaverucha in partial fulfillment of the requirements for the degree of Master of Computer Science.

Dated: July 4, 2006

Supervisor:

Dr. V. Kešelj

Readers:

Dr. K. Johnson

Dr. M. McAllister

DALHOUSIE UNIVERSITY

DATE: July 4, 2006

AUTHOR: Greg Zaverucha

TITLE: Generalized Mersenne Numbers in Pairing-Based Cryptography

DEPARTMENT OR SCHOOL: Computer Science

DEGREE: M.C.Sc.

CONVOCATION: October

YEAR: 2006

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

Signature of Author

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the thesis (other than brief excerpts requiring only proper acknowledgement in scholarly writing) and that all such use is clearly acknowledged.

Table of Contents

List of Tables	vii
List of Figures	ix
Abstract	x
Acknowledgements	xi
Chapter 1 Introduction	1
1.1 Motivation	1
1.2 Goals and Objectives	2
1.3 Thesis Summary	3
1.4 Contributions	5
Chapter 2 Background	6
2.1 Algebraic Background	6
2.1.1 Groups, Rings and Fields	6
2.1.2 Finite Fields	7
2.2 Elliptic Curves	9
2.3 Elliptic Curves Over Finite Fields	10
2.4 Pairings	15
2.4.1 The Weil Pairing	16
2.4.2 The Tate Pairing	17
2.5 Applications of Pairings	18
2.5.1 Relevant Presumably Hard Problems	19
2.5.2 Identity Based Encryption	21
2.5.3 The Menezes-Okamoto-Vanstone Reduction	24
2.5.4 Subgroup Membership	25
2.5.5 An Easy Instance of the ECDDHP	26

Chapter 3	Problem Description	28
3.1	Parameter Sizes and Security Levels	28
3.2	Tate vs. Weil	30
3.3	Quantitative and Practical Issues	32
Chapter 4	Modular Arithmetic	34
4.1	Modular Reduction by General Moduli	34
4.2	Modular Reduction by Special Moduli	36
4.3	Generalized Mersenne Numbers	37
4.3.1	Related Work	43
4.4	Internal Storage of Large Integers	45
4.5	Limb Modular Reduction Weight	46
4.6	A Family of Generalized Mersenne Numbers	48
4.7	Modular Arithmetic Experiments	49
4.8	Results	51
Chapter 5	Pairing Computations	57
5.1	Miller's Algorithm	57
5.1.1	Related work	60
5.2	Pairings with Generalized Mersenne Numbers	65
5.3	Embedding Degree $k = 1$	68
5.3.1	Parameter Search	69
5.3.2	Impracticality of $k = 1$	71
5.3.3	Conclusions for $k = 1$	73
5.4	Embedding Degree $k = 2$	73
5.4.1	Parameter Search	74
5.4.2	Implementation and Experiments	75
5.4.3	Results	77
5.5	Conclusions	79
Chapter 6	Implications for Pairing-Based Cryptography	81
6.1	Time Comparisons	81

6.2	Vulnerability of Generalized Mersenne Primes to the NFS	84
6.3	Recommendations	86
Chapter 7	Conclusion	88
7.1	Future Work and Open Problems	88
Appendix A	List of Notation	90
Appendix B	Test Platform Hardware and Software Particulars . . .	92
Bibliography	93

List of Tables

Table 1.1	The “layers” of a pairing-based cryptosystem. Lower layers are used to implement upper layers.	2
Table 3.1	Minimum bitlengths for n, p^k, q^k . (p is prime, q is a power of 2 or 3. b_{q^k} is the size of \mathbb{F}_{q^k} in bits. Table adopted from [41].)	29
Table 4.1	Table of all primes of the form $p = f(2^{32 \cdot k})$ for $f(t) = t^d - t^{d-1} + 1$, such that $bitlength(p) < 16000$ and $d < 10^4$	49
Table 4.2	Polynomials with a variety of modular reduction weights used. The moduli for benchmarking were set to $f(2^k)$ for all $k \equiv 0 \pmod{32}$ such that $\log_2 f(2^k) < 16000$	51
Table 4.3	Comparison of REDC baseline to MRW 1.	53
Table 4.4	Comparison of REDC baseline to MRW 35.	53
Table 4.5	Comparison of REDC baseline to MRW 63.	53
Table 4.6	Comparison of REDC baseline to MRW 188. “Security Level” displays a range or approximate level when the bitsize of $f(2^k)$ differs significantly from the bitsize required by the security level.	53
Table 5.1	Examples from [41] for embedding degree 1. The exact MRW of example 3 would require a careful implementation to compute, but by partially computing X we were able to determine that it is greater than 10^{25}	69
Table 5.2	List of polynomials $n(t)$ and $h(t)$ such that $n(2^q)$, and $p(2^q)$ are both prime where $p(t) = [n(t)h(t)]^2 + 1$	71
Table 5.3	Examples used in [41] for embedding degree two. At the 80 and 256-bit security levels (Examples 6 and 9) the MRW is low enough to allow practical implementation.	74
Table 5.4	Some polynomials producing primes $n = n(2^q)$ and $p = n(2^q)h(2^q) - 1$, sorted by $MRW(p)$. b_n and b_p are the number of bits in n, p	75
Table 5.5	Parameters used in our implementation.	75
Table 5.6	Miller’s algorithm at the 80-bit security level (MRW 36, $b_p = 512$).	78
Table 5.7	Miller’s algorithm at the 80-bit security level (MRW 44, $b_p = 704$).	78

Table 5.8	Miller’s algorithm at the 128-bit security level.	79
Table 5.9	Miller’s algorithm at the 192-bit security level.	79
Table 5.10	Weil/Tate comparison at the 80-bit security level.	80
Table 5.11	Weil/Tate comparison at the 128-bit security level.	80
Table 5.12	Weil/Tate comparison at the 192-bit security level.	80
Table 6.1	Table 3 from [41], pairing computation cost for each bit of n . An additional row $k = 1^*$ was added for the case when Solinas reduction is <i>not</i> used.	82
Table 6.2	Updated Table 3 from [41]. $T(b)$ replaced with faster of REDC, GMP. $\tilde{T}(b)$ replaced with timing at MRW 15. For $b \not\equiv 0 \pmod{32}$, the nearest larger timing was used (ie. $171 \rightarrow 192$, $1365 \rightarrow 1376$, $683 \rightarrow 704$).	83
Table 6.3	The examples used in [63]. b_p is the size of the field \mathbb{F}_{p^k} in bits. $b_p - \gamma$ is the estimated security in bits provided. The last column shows the percentage decrease in the security provided.	85
Table 6.4	Comparison of modular multiplication times. In examples 4 and 6 the times shown are for multiplication mod a $b_{p^2}/2$ bit prime. We show the timings for 10^4 modular multiplications.	86

List of Figures

Figure 2.1	Geometric interpretation of addition on an elliptic curve over \mathbb{R} .	11
Figure 4.1	Time required to compute $m \pmod{N}$ for 10^6 integers $N < m < N^2$.	47
Figure 4.2	Modular multiplication times for varying MRW, the REDC baseline and the standard GMP library implementations. Most of the MRW plots coincide to form the thick line.	54
Figure 4.3	Modular multiplication times for selected MRW, and the REDC baseline.	55
Figure 4.4	Modular multiplication times for selected MRW and the REDC baseline at bitlevels less than 1500.	56
Figure 5.1	Plot of $\text{MRW}([n(t)t]^2 + 1)$ for $n(t)$ with modular reduction weight from 0 to 100.	72
Figure 5.2	Plot of $\text{MRW}([p(t)t]^2 + 1)$ for $n(t)$ with modular reduction weight from 0 to 50.	73

Abstract

Pairing-based cryptosystems have been the subject of much recent research because of the unique features they provide. As the technology matures, research is looking to the future. Efficient implementation is of greater importance at high-security levels. Generalized Mersenne (GM) numbers can impact performance in two ways. Their structure can be exploited to perform fast modular reduction, a costly part of field arithmetic. The second opportunity to make use of generalized Mersenne numbers is Miller's algorithm, the workhorse of pairing computations. This second use is possible when the elliptic curve used is chosen such that it has a subgroup of order n where n is a low-weight GM prime.

Of particular interest to pairing-based cryptography are elliptic curves which provide both speedups. Due to the limits of known curve construction techniques, only supersingular curves with embedding degree one and two can be created with GM parameters. In the case of embedding degree one, we argue that current construction techniques are impractical. For embedding degree two, we find parameters suitable for implementation and quantify their impact on the computation of the Weil and Tate pairings.

This work is the first to quantify the efficiency gained by arithmetic modulo GM numbers. The moduli used to benchmark GM methods cover a range of sizes and modular reduction weights. A drawback we consider is the increased vulnerability to a variant of the number field sieve. We show that despite requiring larger parameters to maintain security, GM primes can still offer improved performance over general primes. Finally, we use our results to strengthen previous analysis. This puts our work into a larger context, allowing for comparisons and recommendations relevant to cryptographic applications based on pairings.

Acknowledgements

I would like to thank my supervisor, Dr. Kešelj, for his help, support and willingness to take on something different. I also thank the many faculty and peers in both the computer science and math departments of Dalhousie for discussions along the way. The readers of this thesis Dr. Vlado Kešelj, Dr. Keith Johnson and Dr. Mike McAllister were especially helpful, providing feedback and comments. Thanks are due to my parents Pat and Walter who have been there for me since the beginning. They made this possible, especially with logistical support getting me to and from Halifax. Thank you also to Kate, who makes time fly.

Chapter 1

Introduction

In this work, we examine two ways that generalized Mersenne numbers can be used to reduce the time required to compute pairings on elliptic curves. Our goal is to quantify the benefits provided by generalized Mersenne numbers across a broad range of parameter sizes.

1.1 Motivation

In recent years fascinating cryptographic applications of the Tate and Weil pairings have been discovered. Pairings have become the primitive at the center of new cryptosystems which provide unique features. The example which generated the most interest is Identity Based Encryption (IBE). IBE is a public key cryptosystem where an arbitrary string can be used as a public key. For example, a user's email address can be used by the sender to encrypt mail for them. This is much simpler for the sender, who does not need to obtain and verify the authenticity of a public key. To date, the only practical and satisfactory IBE scheme [16] is based on admissible bilinear maps. In practice, the Weil and Tate pairings on elliptic curves provide these maps. After reviewing some related background on elliptic curves in Chapter 2, we discuss IBE in detail, as well as other applications in Section 2.5.

When high security is demanded from pairing based cryptosystems, implementation decisions suitable at lower levels may no longer be optimal, and the choice of parameters must be re-examined. Due to the “layered” nature of pairing-based cryptosystems, there are a large number of these decisions, and tradeoffs arise. The layers of a pairing-based cryptosystem begin with a finite field and corresponding arithmetic. Then follows the elliptic curve group and group arithmetic, then the computation of pairings by Miller's algorithm. These pairings finally get used as building blocks in protocols. The layered complexity of pairing based cryptography is shown in Table 1.1.

5	Pairing-based cryptosystem (IBE, BLS signatures, ...)
4	Tate or Weil pairing
3	Miller's algorithm
2	Elliptic curve group arithmetic
1	Finite field arithmetic (\mathbb{F}_{p^k})

Table 1.1: The “layers” of a pairing-based cryptosystem. Lower layers are used to implement upper layers.

Generalized Mersenne (GM) numbers appear at layers 1 and 3. At the first layer, if p is a GM number (with certain additional properties) then the structure the GM number provides can be exploited to improve the efficiency of reduction modulo p . A significant cost of arithmetic in finite fields is modular reduction, and typically one reduction is performed after every multiplication. Chapter 4 will focus on GM numbers used for field arithmetic.

The other layer at which GM numbers play a role is in Miller's algorithm (layer 3), used to compute both the Tate and Weil pairings. We review the background for Miller's algorithm in Chapter 2 and the algorithm itself in Chapter 5.

1.2 Goals and Objectives

Although using generalized Mersenne numbers for fast modular reduction has been known for some time, limited research has been done to quantify the improvement. It is assumed to be faster than other reduction algorithms, but we cannot say by how much, and do not know how performance changes as the operand sizes increase. There is also a measure called the *modular reduction weight* (§4.3) for a generalized Mersenne number which describes the cost of reduction. We would like to know the practical upper bound for this weight, and determine how reduction slows as it increases. To support these goals, we are also interested in efficient methods to find primes with modular reduction weight below the practical upper bound.

Likewise, many authors recommend the use of GM primes when computing pairings, to allow a variant of Miller's algorithm to take advantage of their structure. This choice reduces the time required to compute the pairing, but again the improvement is not quantified. For instance, it has not been compared to a competing strategy —

choosing a key parameter to be a prime with low Hamming weight.

We also consider the associated problem of finding suitable elliptic curves with parameters which are GM primes. We require a curve over a field of characteristic p , where p is a GM prime, and the order of the curve must also be divisible by a GM prime n . If this curve is to be used for cryptography, then p and n must be large enough to provide adequate security, but not much larger, for the cost of arithmetic also increases with their size.

To date, curves with GM parameters can be chosen in two situations, supersingular curves with embedding degree one and two. However, the construction techniques ignore the modular reduction weight, as well as another property which facilitates practical implementation. We will determine whether these methods produce curves which are useful in practice. This question is addressed in Chapter 5.

We are ultimately interested in the effect these parameters will have on pairing-based cryptography. How important are GM primes, and what characteristics are necessary for them to be practical? Choosing to use GM parameters often involves a tradeoff of competing optimizations. Since the cases where curves can be constructed with GM parameters are limited, we may be forced to forgo other optimizations.

Once the benefits of GM parameters have been quantified, it will be possible to make comparisons when making implementation decisions. In addition, knowing when GM parameters are most effective may motivate further research into curve construction techniques.

There is also the potential that using these parameters reduces the security provided. Cryptanalysis may be adapted to exploit the the same structure that allows fast implementations, presenting a serious threat to security. We take this into consideration when discussing the implications to pairing-based cryptography in Chapter 6.

1.3 Thesis Summary

This thesis spans a broad range of topics required for a comprehensive quantitative evaluation of pairings with generalized Mersenne primes. We have brought together the work of Solinas (modular arithmetic) and Miller (pairing computations) and improved the analysis of Koblitz and Menezes by including new empirical results. Before making recommendations, we also incorporate performance estimates of the number

field sieve. All together, we compare implementation options at a range of security levels. As well as quantifying performance, we identify some practical limitations of generalized Mersenne parameters.

We present timings from multiple implementations of modular reduction functions. We then compare them for a variety of moduli — with different modular reduction weights and sizes. The range of sizes spans the current requirement for pairing-based cryptography, to anticipated future requirements. A hurdle for this large number of comparisons is that each modulus requires a unique implementation of the reduction function, since reduction is based on the structure of the modulus. This was possible due to a new code generation tool, which we developed and used to create all the required modular reduction functions. We identify what performance can be expected at each weight and size, and discover what modular reduction weights are practical. As the weight changes, we will describe the corresponding performance changes. This knowledge of what moduli are practical, is put to use to guide the next part of this work.

We then move on to examine curve construction methods for supersingular curves with embedding degrees one and two. Curves of embedding degree one have been overlooked by researchers, who have favored curves with large embedding degree. Until now, these curves were discarded without good reason. After examining curve construction techniques (which yield GM parameters), we identify issues that frustrate implementation and incorporate them into a search for suitable parameters. Negative results lead us to argue that current methods are not practical. We give the first solid reasons to avoid curves of embedding degree one, supported by a combination of our empirical results and previous analysis.

At embedding degree two, we confirm that curve construction methods can lead to practical implementations. By including some additional constraints, we found parameters well suited to implementation. We describe our implementation and provide timings; again, with a variety of GM parameters over a range of sizes. These timings can also be used as good estimates for the relative speed of low-Hamming weight implementations. This optimization is similarly frequently recommended, but without implementation experiments it can be difficult to assess its importance.

Our arithmetic timings were also used to strengthen previous analysis. With added

knowledge of the performance of GM arithmetic we were able to make comparisons in a larger context. The timings for pairing computation in our implementation also lend credibility to this analysis.

A recently discovered optimization to the number field sieve reduces the time required to solve discrete logarithms in a finite field when the characteristic is a GM prime. Maintaining the difficulty of this problem is necessary for pairing-based cryptosystems. Based on our results, we suggest that this threat is not significant enough to abandon the use of GM primes. When we consider the time savings provided by GM primes, even in a larger field they are still far faster than competing methods.

During the course of this work, we have identified new and interesting open problems. We will list and describe them in the final chapter.

1.4 Contributions

We would like to emphasize the following main contributions of the thesis:

- Quantify the performance of arithmetic modulo GM primes with modular reduction weight from 1-200 and bitsize from 0-16000.
- Examined curve construction techniques for creating curves with GM prime parameters, and found to be inadequate at embedding degree one. At embedding degree two we modified existing methods to produce parameters that have low modular reduction weight, and are word aligned.
- Used knowledge of arithmetic costs to compare implementation options for pairing based cryptography at high security levels.
- Offer the first discussion of the impact of the Schirokauer's NFS on pairing based cryptosystems. Our work suggests that the benefits of arithmetic modulo GM primes outweigh improvements to the NFS.

Chapter 2

Background

This chapter begins with a brief review of some related topics in algebra. We then review elliptic curves, function fields and divisors in §2.2, §2.3 and §2.3. At this point we will have covered the required background to describe the Weil (§2.4.1) and Tate (§2.4.2) pairings. In the last section (§2.5) we will discuss some applications of pairings.

2.1 Algebraic Background

2.1.1 Groups, Rings and Fields

A set G closed under a binary operation $*$ forms a *group* $(G, *)$ when the following properties are satisfied.

1. $*$ is associative; $a * (b * c) = (a * b) * c, \forall a, b, c \in G$.
2. There exists an identity element $e \in G$, such that $a * e = a = e * a$ for all $a \in G$.
3. Inverses exist for all elements, $a * a^{-1} = e = a^{-1} * a$ for all $a \in G$.

When the operation $*$ is also commutative, i.e. $a * b = b * a$, for all $a, b \in G$, we call G an *abelian* group. The groups of interest to us will always be abelian, and we will often write “group” in place of “abelian group”.

There are two concepts that will prove useful when dealing with finite groups. First, we have the order of $a \in G$, which is the smallest positive integer n such that

$$a^n = \underbrace{a * a * \dots * a}_{n \text{ times}} = e .$$

The accompanying notation is $\text{ord}(a) = n$. Equivalently this can be defined as $\text{ord}(a) = |\langle a \rangle|$, where $\langle a \rangle$ is the subgroup of G generated by a . The *group order*, is simply the number of elements in G , denoted $|G|$. A famous theorem of Lagrange

states that $|H|$ divides $|G|$ for any subgroup $H \subseteq G$. This implies that the $\text{ord}(a)$ divides $|G|$.

Second, for a finite group G , the *exponent* of G , denoted $\exp(G)$, is the smallest integer n such that $g^n = 1$ for all $g \in G$. If G is abelian, then there exists $g \in G$ such that $\text{ord}(g) = \exp(G)$.

In some cases a set R is closed under two operations, an addition (written $a + b$) and a multiplication (written $a \cdot b$ or ab). If this is the case, $(R, +, \cdot)$ is a commutative *ring* with unity if

1. $(R, +)$ is an abelian group (with identity element 0),
2. (R, \cdot) is a monoid, i.e. multiplication is a valid operation with identity element 1 (called the *unity*),
3. multiplication is commutative,
4. and multiplication distributes over addition, i.e. $a(b + c) = ab + ac = (b + c)a$.

An *ideal* I of R is an additive subgroup such that for all $r \in R, s \in I$ their product $rs \in I$.

A *field* is a set K closed under an addition operation and a multiplication operation such that $(K, +)$ and $(K - \{0\}, \cdot)$ are abelian groups. Examples of common infinite fields are \mathbb{C}, \mathbb{R} and \mathbb{Q} the fields of complex, real and rational numbers. Fields can also be finite, for example the integers mod p , where p is a prime. The following section reviews these fields in greater detail. In what follows, when the particular field is not important we will refer to it as “a field K ”.

A *ring of polynomials*, denoted $K[x_1, \dots, x_n]$, is the set of all polynomials in indeterminates x_1, \dots, x_n with coefficients in the field K , and the familiar polynomial arithmetic operations. As an example, $\mathbb{R}[x, y]$ is the ring of polynomials in x, y with real coefficients, one element being $x^2 + 3y^2 + \sqrt{19}$.

2.1.2 Finite Fields

For the applications we will discuss, the field K is chosen to be finite. The fields used are the Galois fields, written $GF(p^m)$, or \mathbb{F}_{p^m} where p is prime and $m \in \mathbb{N}$.

To define Galois fields, there are two cases, when $m = 1$ and when $m > 1$. In the prime field case ($m = 1$), we have the integers modulo p , $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = \mathbb{Z}_p$. Galois fields with $m > 1$ are called *extension fields*. We will describe them shortly. For cryptographic applications the field $GF(2^m)$ is common because it can be represented in a way that allows fast arithmetic using basic operations. A modern reference on the subject of arithmetic and representation in finite fields for elliptic curve applications is [38]. More general references on the subject are [25], Chapter 9 and [39], Chapter 4. Unless stated otherwise, we will write \mathbb{F}_q for the field with $q = p^m$ elements.

The *characteristic* of a field K is the least integer c such that $c \cdot a = 0$ for all $a \in K$. For example \mathbb{F}_{2^m} has characteristic 2, \mathbb{F}_{101} has characteristic 101, and in general \mathbb{F}_{p^m} has characteristic p .

Extension Fields

We start with an example of an extension field. Consider the polynomial $x^2 + \alpha$ in the ring $\mathbb{F}_p[x]$, and further suppose that $-\alpha$ is not a quadratic residue mod p . Then $x^2 \equiv -\alpha \pmod{p}$ has no solution in \mathbb{F}_p , and the roots $\pm\sqrt{-\alpha}$ are not in \mathbb{F}_p . To remedy this situation we will *adjoin* an element θ to the field such that $\theta^2 = -\alpha$ and call this new larger field $\mathbb{F}_p(\theta)$. The general form of elements in this field is $a + b\theta$ for $a, b \in \mathbb{F}_p$. The larger field has p^2 elements. By adjoining θ , we have created \mathbb{F}_{p^2} , an extension of \mathbb{F}_p with degree 2. This field is isomorphic to the quotient group $\mathbb{F}_p[x]/\langle x^2 + \alpha \rangle$, where $\langle x^2 + \alpha \rangle$ is the ideal of $\mathbb{F}_p[x]$ generated by $x^2 + \alpha$. The result of repeatedly adjoining elements is often referred to as a *tower of fields* or a *tower of extensions*.

In general, extension fields can be created using

$$\mathbb{F}_{p^m} = \frac{\mathbb{F}_p[x]}{\langle p(x) \rangle} \quad \text{where } p(x) \in \mathbb{F}_p[x] \text{ is a monic irreducible polynomial of degree } m$$

in a way analogous to $\mathbb{Z}/p\mathbb{Z}$, except instead of reduction mod p we reduce polynomials mod $p(x)$ using the division algorithm in $\mathbb{F}_p[x]$. This variation of Euclid's classic algorithm is described in §2.2 of [25].

The field \mathbb{F}_{p^m} is an extension of degree m of \mathbb{F}_p . In general, if E is an extension of F , the *degree* of this extension, $[E : F]$, is the degree of E as a vector space over F . To clarify this, we generalize our example. Elements of a degree m extension will look

like $a_0 + a_1\theta_1 + \dots + a_{m-1}\theta_{m-1}$, and we can represent elements as $(a_0, \dots, a_m) \in F^m$ with respect to a basis $\{1, \theta_1, \dots, \theta_{m-1}\}$.

Algebraic Closure

A field K is *algebraically closed* if every polynomial in $K[x]$ has a zero in K . Equivalently, this can be defined in terms of the factorization of polynomials.

Theorem 1 *A field K is algebraically closed iff every nonconstant polynomial in $K[x]$ can be written as a product of linear factors in $K[x]$.*

See [32], §31 for a proof. An algebraic closure exists for every field, and we write it \overline{K} . Any algebraically closed field is infinite¹. The field \mathbb{F}_q is not algebraically closed; and $\overline{\mathbb{F}_q}$ is an infinite field.

2.2 Elliptic Curves

In this section we review necessary material related to elliptic curves. References covering this material in greater detail are [19, 20, 70, 71].

Let K be a field. An *elliptic curve* E over K (denoted E/K) is a non-singular cubic curve given by

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad (2.1)$$

where $a_i \in K$. This standard form is called the long Weierstrass form, but often² in applications $a_1 = a_2 = a_3 = 0$ and the shorter form:

$$y^2 = x^3 + ax + b \quad (2.2)$$

($a, b \in K$) is used instead. In order for the curve (2.2) to be non-singular, the discriminant $\Delta = 4a^3 + 27b^2$ must be nonzero. (For an equivalent condition for (2.1) see [70] p. 46–50.) We are most interested in the set of points

$$E(K) = \{(x, y) \in K^2 : y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}.$$

¹Exercise 85 of the *Supplementary Material* section of [69], available from <http://shoup.net/ntb>

²Primarily when $\text{char}(K) \neq 2, 3$.

$E(K)$ contains one extra point \mathcal{O} , called the *identity* or the *point at infinity*. In the affine model, this point is simply represented $\mathcal{O} = (\infty, \infty)$ however in projective coordinates $[X : Y : Z]$, where $x = X/Z$, $y = Y/Z$ we can write $\mathcal{O} = [0, 1, 0]$.

Naming \mathcal{O} the identity suggests a group structure, and indeed this is the case. The group operation is written additively. For points $P_i = (x_i, y_i) \in E(K)$ (where $E(K)$ is of the form (2.2)) the sum $P_3 = P_1 + P_2$, can be computed as follows.

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda x_3 + y_1 - \lambda x_1 = \lambda x_3 + y_2 - \lambda x_2 \end{aligned}$$

where the slope, λ is given by

$$\lambda = \frac{y_1 - y_2}{x_1 - x_2}$$

when $P_1 \neq P_2$ and

$$\lambda = \frac{3x^2 + a}{2y}$$

when $P_1 = P_2$. The quantity λ is called the slope because the group laws are derived geometrically. Many introductory texts cover this derivation in detail, we simply include Figure 2.1 as a reminder. See the references given at the beginning of this section for the group operation when the curve is in another form. For arithmetic in projective coordinates (and variations thereof) see [38].

As suggested above, \mathcal{O} is the identity, and the inverse of a point $P = (x, y)$ is defined³ to be $-P = (x, -y)$. There is no multiplication operation in this group, but a “scalar multiplication” is frequently used. This allows a point to be “multiplied” by an integer, written $[m]P$ (or sometimes just mP) which amounts to repeated addition

$$[m]P = \underbrace{P + P + P + \dots + P}_{m \text{ times}} .$$

Since this operation is analogous to exponentiation, it enjoys all of the existing fast methods allowing computation in $O(\log m)$ point additions.

2.3 Elliptic Curves Over Finite Fields

Let K be a finite field $K = \mathbb{F}_q$. The group $E(K)$ is then finite, and the field elements have a bounded size, facilitating elliptic curve arithmetic with computer hardware.

³For curves in Weierstrass form.

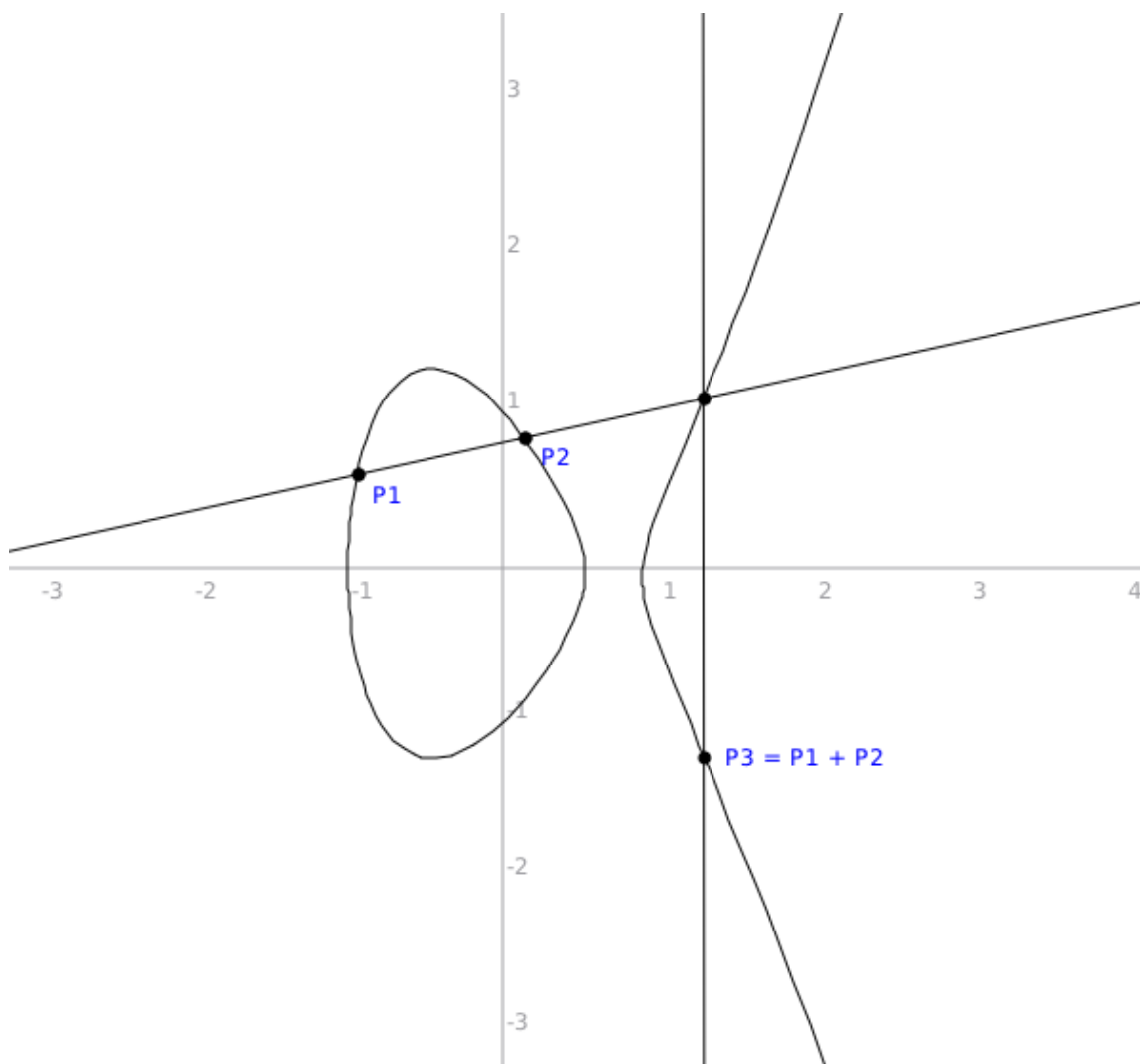


Figure 2.1: Geometric interpretation of addition on an elliptic curve over \mathbb{R} .

Point Counting

When working with $E(\mathbb{F}_{p^m})$ the group order is finite and often required by applications. We use $\#E(\mathbb{F}_{p^m})$ to denote the group order of E . An elegant theorem, due to Hasse bounds the number of possible solutions to (2.1).

Theorem 2 *For an elliptic curve E defined over \mathbb{F}_q ,*

$$\#E(\mathbb{F}_q) = q + 1 - t \quad \text{where} \quad |t| \leq 2\sqrt{q}$$

A proof of the theorem is available in [70], §V.1 as well as [20] page 50. The integer t is called the *trace* of the curve. In §2.3 we will classify a family of curves based on their trace.

While once a difficult problem, recent advances in point counting algorithms have been dramatic. It is now practical to determine the group order of much larger curve groups than those used in practical cryptographic applications. Further, most applications only require that the group order be computed once at the outset, which can be done offline. Interested readers should consult [28] Chapters VI - *Determining the Group Order* and VII - *Schoof's Algorithm and Extensions*, as well as [29] Chapter IV *Advances in Point Counting*. These sources also include extensive references on this subject. Most symbolic calculators (e.g. PARI/GP, Maple, Mathematica) and programming libraries supporting computations with elliptic curves (e.g. LiDIA, MIRACL) include implementations of point counting algorithms suitable for our purposes.

The n -torsion subgroup

As is the case in all finite groups, the smallest positive integer n such that $[n]P = \mathcal{O}$ is called the *order* of a point. The n -torsion subgroup $E[n]$ is the subgroup of points having order dividing n , i.e.

$$E[n] = \{P \in E : [n]P = \mathcal{O}\}$$

Over an algebraically closed field, $E(\overline{K})$ has n^2 points, and the group structure $\mathbb{Z}_n \times \mathbb{Z}_n$ ([70], §III.4). In the case of non-algebraically closed fields, a theorem of Balasubramanian and Koblitz [10] describes $E[n]$.

Theorem 3 *Let E be an elliptic curve over \mathbb{F}_q , where $q = p^m$. Let n be a positive integer dividing $\#E(\mathbb{F}_q)$ and not dividing $(q - 1)$. Then the n -torsion subgroup of $E(\mathbb{F}_{q^k})$ has size n^2 if and only if $n \mid (q^k - 1)$.*

Supersingular Curves

Supersingular elliptic curves are an important family of curves in cryptographic applications because of some of the properties they possess. There are multiple equivalent definitions for a curve to be supersingular. The one that follows is most useful in this work.

Definition 1 *An elliptic curve $E(\mathbb{F}_q)$, $q = p^m$, with order $\#E = p + 1 - t$ is said to be supersingular iff*

$$t^2 \in \{0, q, 2q, 3q \text{ or } 4q\}$$

This definition is a refinement, obtained by combining the more general definition: E/K is supersingular if $\text{char}(K) \mid t$, with some results on the existence of curves (see [49]). For some other, more theoretical definitions of supersingular curves see [70], §V.3. Supersingular elliptic curves can be classified into six classes according to their trace⁴.

Function Fields

We will need a few pieces in order to build up a function field. For this section, let E/K be an elliptic curve in short Weierstrass form (2.2). The end result will be a field of rational functions related to a particular curve.

First consider the polynomial ring $K[x, y]$. We can limit its size by considering the *coordinate ring*

$$K[E] = \frac{K[x, y]}{\langle y^2 - x^3 - ax - b \rangle}.$$

This equates polynomials that differ by a multiple of the equation of the curve. It is still an infinite ring of polynomials, but their degree is bounded. To create a *function field*, consider the fractions

$$K(E) = \frac{a(x, y)}{b(x, y)} \quad a(x, y), b(x, y) \in K[E].$$

The elements of $K(E)$ are often referred to as *rational functions*.

⁴For a table, see [49].

We can represent all the rational functions $r \in K(E)$ (that are not identically zero) in a uniform way with respect to a point $P \in E$. There exists a rational function u_P with $u_P(P) = 0$ such that

$$r = u_P^d s, \quad d \in \mathbb{Z}, \quad s \in K(E), s(P) \neq 0$$

The rational function u_P is called a *uniformizer* at P . The *order of a rational function f at a point $P \in E$* , denoted $\text{ord}_P(f)$ is the integer d .

Divisors

A *divisor* is a formal sum of points;

$$\sum_{P \in E} m_P(P) = m_{\mathcal{O}}(\mathcal{O}) + m_{P_1}(P_1) + m_{P_2}(P_2) + \dots$$

with a finite number of nonzero $m_P \in \mathbb{Z}$. The set of points having nonzero m_P is called the *support* of a divisor. An addition operation for divisors is defined for two divisors \mathcal{D}_1 and \mathcal{D}_2 as follows:

$$\mathcal{D}_1 + \mathcal{D}_2 = \sum_{P \in E} m_P(P) + \sum_{P \in E} n_P(P) = \sum_{P \in E} (m_P + n_P)(P) \quad (2.3)$$

Under addition of divisors, the set of all divisors forms a group, denoted $\text{Div}(E)$. The degree of a divisor $\mathcal{D} \in \text{Div}(E)$ is

$$\text{deg}(\mathcal{D}) = \sum_{P \in E} m_P$$

The divisors with degree zero form an important subgroup, denoted $\text{Div}^0(E)$. To every rational function f , the associated divisor is given by

$$\text{div}(f) = \sum_{P \in E} \text{ord}_P(f)(P).$$

The converse however, that every divisor can be associated to a rational function, is not always true. Divisors that actually “belong” to some rational function are called *principal* divisors. $\text{Prin}(E)$ denotes the set of principal divisors, and

$$\begin{array}{ccccccc} \text{Prin}(E) & \subseteq & \text{Div}^0(E) & \subset & \text{Div}(E) \\ \text{principal divisors} & \subseteq & \text{divisors of degree 0} & \subset & \text{all divisors} . \end{array}$$

For our purposes, one further refinement is necessary. To work with the divisors of degree zero that **are not** principal, we define the *the divisor class group*⁵:

$$\text{Pic}(E) = \text{Div}(E)/\text{Prin}(E)$$

and the *degree zero part of the divisor class group* or *Jacobian*:

$$\text{Pic}^0(E) = J(E) = \text{Div}^0(E)/\text{Prin}(E) .$$

Two divisors \mathcal{D}_1 and \mathcal{D}_2 are in the same coset, or in the same *divisor class* if $\mathcal{D}_1 - \mathcal{D}_2$ is principal (\mathcal{D}_1 and \mathcal{D}_2 may also be called *linearly equivalent*). Equivalence is denoted $\mathcal{D}_1 \sim \mathcal{D}_2$. Every divisor class has a unique representative of the form $(P) - (\mathcal{O})$ for all $P \in E$ giving a one-to-one correspondence between the Jacobian and the points on the curve, given by

$$\begin{aligned} \phi : E &\rightarrow J(E) \\ \phi(P) &= (P) - (\mathcal{O}) \end{aligned}$$

One can make use of ϕ and the fact that divisor addition (2.3) is trivially associative to give an elegant proof of the associative law in $E(K)$.

A function f can be evaluated at a divisor $\mathcal{D} = \sum_{P \in E} m_P(P)$ as follows

$$f(\mathcal{D}) = \prod_{P \in E} f(P)^{m_P}$$

When f and \mathcal{D} are defined over a common field K , the product $f(\mathcal{D}) \in K$ as well.

2.4 Pairings

First we will explain the general characteristics of a mapping that make it a pairing, then we will look at two specific pairings of importance to cryptography in §2.4.1 and §2.4.2.

Bilinear Mappings

For groups G_1, G_2, G_3 , where G_1 and G_2 have exponent n and G_3 cyclic of order n , a *pairing* is a map:

$$e : G_1 \times G_2 \rightarrow G_3 ,$$

⁵We make two remarks:

i) the divisor class group is sometimes also called the Picard group, which explains the notation,
 ii) note the similarity to the class group in an algebraic number field which is all fractional ideals modulo all principal fractional ideals.

which satisfies the following two conditions:

1. **Bilinearity:** For all $P, P' \in G_1$ and $Q, Q' \in G_2$

$$e(P + P', Q) = e(P, Q)e(P', Q)$$

and

$$e(P, Q + Q') = e(P, Q)e(P, Q') .$$

Bilinearity can be thought of as a generalized multiplication that satisfies the distributive law.

2. **Non-Degeneracy:** For all $P \in G_1$, $P \neq 0$ there is some $Q \in G_2$ such that $e(P, Q) \neq 1$. Likewise, for all $Q \in G_2$, $Q \neq 0$ there is some $P \in G_1$ such that $e(P, Q) \neq 1$. A non-degenerate map is one that “does something”, it does not simply map everything to the identity of G_3 .

3. **Alternating:** $e(P, P) = 1$, and so $e(P, Q) = e(Q, P)^{-1}$

In the terminology of [16], a bilinear map that is also efficiently computable is called an *admissible* bilinear mapping. In this work all bilinear mappings discussed will be efficiently computable, hence admissible.

2.4.1 The Weil Pairing

We define the *n-th roots of unity* of a field K as

$$\mu_n = \{x \in \overline{K} \mid x^n = 1\}$$

which forms a group under multiplication. There are multiple ways the Weil pairing can be defined [20, 28, 53, 70]. We present the definition that most easily lends itself to computation of this pairing, Proposition 8 of [53].

Definition 2 *Let $E(K)$ be an elliptic curve, and $P, Q \in E(K)$. The Weil pairing is a bilinear mapping*

$$e_n : E[n] \times E[n] \rightarrow \mu_n$$

$$e_n(P, Q) = (-1)^n \frac{f_{n,P}(Q)}{f_{n,Q}(P)}, \quad P \neq Q$$

where $f_{n,P}$ and $f_{n,Q}$ are rational functions in $K(E)$ with divisors

$$\operatorname{div}(f_{n,P}) = n(P) - n(\mathcal{O})$$

$$\operatorname{div}(f_{n,Q}) = n(Q) - n(\mathcal{O})$$

We will use the notation $f_{n,P}, f_{n,Q}$ throughout this work. When $K = \mathbb{F}_q$, the n -th roots of unity belong to a finite extension of \mathbb{F}_q , i.e. $\mu_n \subset \mathbb{F}_{q^k}$. The integer k is called the *embedding degree*, or sometimes the *security multiplier*. The field \mathbb{F}_{q^k} is the smallest extension of \mathbb{F}_q which contains μ_n .

2.4.2 The Tate Pairing

First we create a quotient group from the field K . Let K^* be the multiplicative group of K (i.e. $K^* = K - \{0\}$). Define

$$(K^*)^n = \{a^n : a \in K\} .$$

$(K^*)^n$ is a subgroup of K^* , so the group $K^*/(K^*)^n$ exists. The groups $K^*/(K^*)^n$ and μ_n are isomorphic.

Example. Let $K = \mathbb{F}_7$, $n = 3$.

$$\begin{aligned} \mathbb{F}_7^* &= \{1, 2, 3, 4, 5, 6\} \\ (\mathbb{F}_7^*)^3 &= \{1, 6\} \\ \mathbb{F}_7^*/(\mathbb{F}_7^*)^3 &= \{1\{1, 6\}, 2\{1, 6\}, 3\{1, 6\}, 4\{1, 6\}, 5\{1, 6\}, 6\{1, 6\}\} \\ &= \{\{1, 6\}, \{2, 5\}, \{4, 3\}\} \\ \mu_3 &= \{x \in \mathbb{F}_7 \mid x^3 = 1\} \\ &= \{1, 2, 4\} \end{aligned}$$

with $\phi : \mathbb{F}_7^*/(\mathbb{F}_7^*)^3 \rightarrow \mu_3$ given by $\phi(a(\mathbb{F}_7^*)^3) = a$.

$K^*/(K^*)^n$ can also be thought of as an equivalence relation on K where $r \sim s$ for $r, s \in K$ if $rs^{-1} \in (K^*)^n$.

Now we repeat a similar process on the elliptic curve group. Define

$$nE(K) = \{[n]P : P \in E(K)\} ,$$

which is a subgroup of $E(K)$ with exponent n . Again, we create the quotient group $E(K)/nE(K)$, which also has exponent n . We can describe this group as an equivalence relation on $E(K)$ where two points are related if $P_1 - P_2 \in nE(K)$. The groups

$E(K)/nE(K)$ and $E[n]$ have the same number of points, and for certain supersingular curves, $E[n]$ can be used as representatives (see Theorem IX.22 [29], Lemma 3 [49]).

The Tate pairing is a mapping:

$$\langle \cdot, \cdot \rangle : E(K)[n] \times E(K) \rightarrow K^*/(K^*)^n$$

Choose points $P \in E[n]$ and $Q \in E(K)$. Think of Q as a representative for a class of $E(K)/nE(K)$. The Tate pairing is defined

$$\langle P, Q \rangle_n = f_{n,P}(D_Q)$$

where $f_{n,P}$ is a function such that $\text{div}(f_{n,P}) = n(P) - n(\mathcal{O})$, and D_Q be a degree zero divisor equivalent to $(Q) - (\mathcal{O})$.

Further, the support of D_Q must be disjoint from the support of $\text{div}(f_{n,P})$. Since the supports of $\text{div}(f_{n,P})$ and D_Q are disjoint, $f_{n,P}(D_Q) \neq 0$, and so $f_{n,P}(D_Q) \in K^*$. The quantity $f_{n,P}(D_Q)$ should however, be interpreted as an element of $K^*/(K^*)^n$, which are equivalence classes. An equivalent definition of the Tate pairing simplifies it to $f_{n,P}(Q)$, this will be discussed in §5.1.1.

2.5 Applications of Pairings

We now turn to some of the applications of pairings in cryptography. We will present two cryptographic applications, one destructive, one constructive. The MOV reduction was one of the first applications of pairings to cryptography; it reduces the discrete logarithm problem in the group of points on a supersingular elliptic curve to the discrete log problem in $\mu_n \in \mathbb{F}_{q^k}$, where it is easier to solve (in certain cases). Later, the identity based encryption (IBE) scheme of Boneh and Franklin [16] used pairings to create a cryptosystem that allowed an arbitrary string to be used as the public key.

Before discussing applications, we will describe a few of the computational problems commonly used to build cryptosystems.

2.5.1 Relevant Presumably Hard Problems

The Discrete Logarithm Problem (DLP)

The discrete logarithm problem has been used as the underlying hard problem in various cryptosystems. The most notable are the Diffie-Hellman key exchange [27] and ElGamal encryption/signature algorithms [30, 31].

In a general finite group $(G, *)$ the problem is the following: Given group elements g and h find $l \in \mathbb{Z}$ such that

$$h = l(g) = \underbrace{g * g * \dots * g}_{l \text{ times}} .$$

(Further, g and h are chosen such that l always exists. When G is cyclic, this is true for any two elements.)

Originally for cryptography, G was chosen to be \mathbb{F}_p^* . In this case the problem is written multiplicatively as: Given $\alpha, \beta \in \mathbb{F}_p$, find the integer $l \in [1, p - 2]$ such that $\alpha \equiv \beta^l \pmod{p}$. Extension fields $\mathbb{F}_{p^k}^*$ are also widely used, especially of characteristic two. Methods for solving the DLP in finite fields are faster than in an elliptic curve group, requiring the size of cryptographic parameters to be larger. For a survey, see [47, 55].

The Elliptic Curve Discrete Logarithm Problem (ECDLP)

Since the DLP is defined in terms of general groups, the group of points $E(\mathbb{F}_{p^k})$ of an elliptic curve E/\mathbb{F}_{p^k} is no exception. This was the insight that led Koblitz [40] and Miller [52] to suggest they be used in cryptography. In this case l must be recovered such that

$$Q = [l]P = \underbrace{P + P + \dots + P}_{l \text{ times}}$$

where the points P and Q are given, and the addition above is the group operation on the curve. Elliptic curve groups seem to lack the properties of finite fields that allow for efficient algorithms. When combined with the higher cost of group operations in the group, this allows cryptographic applications to provide higher security with smaller parameter sizes. This increases efficiency while maintaining security.

Certain classes of elliptic curves admit properties that allow efficient calculation of ECDLs. To cover them in detail is beyond our current scope. We simply list them:

- **The MOV Attack.** For an elliptic curve E over \mathbb{F}_{p^m} , this attack replaces the ECDL with an ordinary DL in the field $\mathbb{F}_{(p^m)^k}$. This DL is only easier for small k . If E is supersingular, then $k \leq 6$ and the attack may provide a speedup. Complete details appeared in [49], a condensed description follows in §2.5.3.
- **Weil Descent Attacks.** These methods compute elliptic curve discrete logs when E is taken over a non-prime field (i.e. an extension field). The expected reduction in security declines as the degree of the extension increases, for degrees larger than 128 the attack is ineffective. Hence the binary extension fields \mathbb{F}_{2^m} are safe (for most m of cryptographic size). These attacks are covered in detail (and well referenced) in Chapter 8 of [29].
- **The Anomalous Attack.** This attack will solve the ECDLP in $O(\log p)$ group operations, but only works in subgroups of order $n = p$ where p is the characteristic of the field. If E has trace 1, i.e. $\#E = p + 1 - t = p$, then all discrete logs are easy. For the subgroup case, see [65], for the $\#E = p$ case see [61, 72].

For general curves, no algorithm is known to solve the ECDLP efficiently. In [68] it is shown that a general algorithm, one that does not exploit a particular representation of the group elements has a lower bound of $\Omega(\sqrt{n})$ group operations. These algorithms work in all finite groups in $O(\sqrt{n})$ time and are the best known for solving the ECDLP. Most square-root algorithms for computing logarithms are variations of Pollard's Rho [59] and kangaroo [60] algorithms, or Shank's baby-step giant-step [67]. For a survey, see [75]. They may also be efficiently parallelized to m processors with a linear speedup; van Oorschot and Wiener give a $O(n^{1/2}/m)$ algorithm in [76].

The Diffie-Hellman Problem (DHP)

The DHP for a group G comes in two flavors. Let g be an element of G and u, v integers in $\{1 \dots |G|\}$. The *computational DHP* is to compute g^{uv} when given only g^u and g^v . The *Diffie-Hellman assumption* is that the computational DHP is hard. It is often used in proofs of the security of a cryptosystem.

A closely related variant is the *decision DHP*. In this case we are given $g^x, g^y, g^z \in G$, the problem requires deciding whether $g^x = g^{yz}$. If the decision DHP is assumed hard, this implies the computational DHP is hard as well.

It is known that the DHP can be reduced to the DLP in polynomial time in all groups. Assuming the DLP were easy, it could be used to solve the DHP above; simply compute the integers u, v , then g^{uv} . The converse however, is only known to be true for some groups. The DLP could be solved using a polynomial number of calls to an efficient DHP algorithm if the group order meets the conditions given in [48]. This equivalence was viewed as strong evidence that the two problems were equivalent, which was reassuring for cryptographers since less was known about the DHP while most feel the DLP is difficult. The problem of solving the DLP given an algorithm for the DHP is also called the *gap DHP* [29].

The Elliptic Curve Diffie-Hellman Problem (ECDHP)

The ECDHP is simply the DHP in the group $E(K)$. Given the points $P, [a]P, [b]P \in E(K)$, find the point $[ab]P \in E(K)$. The decision ECDHP is to decide whether $[a]P = [bc]P$.

The Bilinear Diffie-Hellman Problem (BDHP)

This problem arose during the study of bilinear pairings. Since the only groups used to implement pairings in practice are elliptic curve groups, we will use their notation. The BDHP is: given $P, [r]P, [s]P, Q$ in $E(K)$, such that $\zeta = e(P, Q) \neq 1$, compute ζ^{rs} . (Here we use $e(\cdot, \cdot)$ to denote either the Tate or Weil pairing. Note that by bilinearity $e([r]P, [s]P) = e(P, P)^{rs}$, but the problem asks for $e(P, Q)^{rs}$.)

The BDHP is closely related to the ECDHP. The relation is due to the fact that $\zeta^{rs} = e([rs]P, Q)$. Thus, computing $[rs]P$ (from $[r]P, [s]P$) gives ζ^{rs} immediately. However the converse is not known, and the BDHP may be easier than the DHP. To date the only known way of solving the BDHP is by computing discrete logs [41].

2.5.2 Identity Based Encryption

Identity based encryption (IBE) was an idea of Shamir's in 1984 [66]. An identity based cryptosystem is a public key system where an arbitrary string can be used as

the public key. One application is email. Alice can encrypt a message for Bob using his email address as the key. Then Bob contacts the Private Key Generator (PKG), authenticates himself, and obtains his private key to decrypt the message.

An IBE scheme is given by 4 algorithms:

1. Setup: generates system parameters and a master key.
2. Extract: uses the master key to generate the private key corresponding to an arbitrary ID string
3. Encrypt: encrypts messages using the public key ID
4. Decrypt: decrypts messages using the corresponding private key

Between 1984 and 2001 many IBE schemes for signatures and authentication were proposed but were not practical or not satisfactory. In 2001 D. Boneh and M. Franklin proposed an IBE scheme that was practical and satisfactory, based on bilinear maps of groups. They also gave an example of their scheme using the Weil pairing [16].

IBE from Pairings

The paper [16] gives multiple versions of their IBE scheme. We present the four algorithms of **BasicIdent**, the simplest scheme. Although it is not secure against an adaptive chosen ciphertext attack, it clearly illustrates the use of pairings.

1. Setup

- Generate groups $\mathbb{G}_1, \mathbb{G}_2$ of prime order q , an admissible bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ and choose a generator $P \in \mathbb{G}_1$.
- Randomly choose the master key $s \in \mathbb{Z}^+$
- Choose cryptographic hash functions;
 $H_1 : \{0, 1\}^* \rightarrow \mathbb{G}_1^*$ and
 $H_2 : \mathbb{G}_2 \rightarrow \{0, 1\}^n$ (n is the bitlength of plaintext messages)
- The message space is $\mathcal{M} = \{0, 1\}^n$ and
the ciphertext space space is $\mathcal{C} = \mathbb{G}_1^* \times \{0, 1\}^n$
- set $P_{pub} = [s]P$

- **params** = $[q, \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_{pub}, H_1, H_2]$ is made public, while s is kept secret.

2. Extract

- For a bit string **ID**, compute $Q_{ID} = H_1(ID) \in \mathbb{G}_1^*$ and set the private key $D_{ID} = [s]Q_{ID}$

3. Encrypt ⁶

- $Q_{ID} = H_1(ID) \in \mathbb{G}_1^*$
- Choose random $r \in \mathbb{Z}_q^*$
- Compute $g_{ID} = e(Q_{ID}, P_{pub}) \in \mathbb{G}_2^*$,
- The ciphertext $C = [rP, M \oplus H_2((g_{ID})^r)]$

4. Decrypt

- For ciphertext $C = [U, V] \in \mathcal{C}$ encrypted with **ID**

$$M = V \oplus H_2(e(D_{ID}, U))$$

where $D_{ID} \in \mathbb{G}_1^*$ is the private key

Let us review why this works. When a message M is encrypted, it is XORed with the hash of $e(Q_{ID}, U)$. Decryption XORs the ciphertext V with the hash of $(g_{ID})^r = e(D_{ID}, P_{pub})^r$. We can show these are the same by using the bilinearity property of pairings.

$$e(D_{ID}, U) = e([s]Q_{ID}, [r]P) = e(Q_{ID}, P)^{sr} = e(Q_{ID}, P_{pub})^r$$

Suppose an eavesdropper had access to ciphertext and the public system parameters. Their challenge is to compute the decryption mask $e(Q_{ID}, P_{pub})^r$ from $P_{pub} = [s]P$, Q_{ID} , P , $[r]P$. This problem is related to the BDHP [29]. They could also try to recover the master key s by solving an instance of the ECDLP since they have both P and P_{pub} .

⁶In this section we use the symbol \oplus to denote the bitwise exclusive OR operation. We also write “XOR” for this operation.

2.5.3 The Menezes-Okamoto-Vanstone Reduction

The *MOV reduction* reduces the elliptic curve discrete logarithm problem (ECDLP) to the discrete logarithm problem (DLP) in a finite field. It originally appeared in [49] using the Weil pairing, however it was later shown that it could also be done using the Tate pairing [33]. We will follow [49]. This problem is relevant to the security of elliptic curve cryptosystems, as their security depends on the difficulty of the ECDLP. When given two points $P, Q \in E(K)$, such that $Q = [\lambda]P$ for some $\lambda \in \mathbb{Z}$, the ECDLP is to recover λ .

The basic idea is the following. Establish an isomorphism between the subgroup $\langle P \rangle$ of order n , and the n -th roots of unity of \mathbb{F}_{p^k} . Then, provided k is small enough, solve the DLP in the finite field with a subexponential time algorithm such as the index calculus [50] or the function field sieve [5]. For supersingular curves, $k \leq 6$; this is small enough for the dual problem to be tractable. However in general, it is not practical since k is exponentially large. It was shown in [10] that the probability of randomly choosing a non-supersingular elliptic curve with $k \leq \log^2 p$ is negligible. Algorithm 1 is the algorithm as presented in [29].

Algorithm 1 The MOV reduction algorithm.

INPUT: $P, Q \in E(\mathbb{F}_q)$ of prime order n such that $Q = \lambda P$ for unknown λ .

OUTPUT: Discrete logarithm λ of Q to the base P

- 1: Determine smallest k such that $n \mid (q^k - 1)$
 - 2: Choose $S \in E(\mathbb{F}_{q^k})$ with $e(P, S) \neq 1$ (choose randomly; low probability that $e(P, S) = 1$)
 - 3: $\alpha = e(P, S)$
 - 4: $\beta = e(Q, S)$
 - 5: Find λ such that $\alpha^\lambda = \beta$ in \mathbb{F}_{q^k} .
 - 6: **return** λ
-

To show why the algorithm works, we need to use the bilinearity property of

pairings.

$$\begin{aligned}
\alpha &= e(P, S) \\
\beta &= e(Q, S) \\
&= e([\lambda]P, S) \\
&= e(P, S)^\lambda = \alpha^\lambda
\end{aligned}$$

This algorithm runs in time exponential in k , since the DLP in \mathbb{F}_{q^k} has runtime exponential in k . Following the discovery of this reduction, implementation of elliptic curve cryptosystems avoided using supersingular curves.

2.5.4 Subgroup Membership

The subgroup membership problem in an abelian group is to decide whether two group elements are members of the same (proper) subgroup. In [53] Miller gives a result that allows us to efficiently make this decision for elliptic curve groups and their torsion subgroups.

Theorem 4 *For two points $P, Q \in E[n]$, $Q \in \langle P \rangle$ if and only if $e(P, Q) = 1$.*

Proof. First note that if $Q \in \langle P \rangle$, then $Q = [l]P$ (for some $l \in \mathbb{Z}$) and

$$e(P, Q) = e(P, [l]P) = e(P, P)^l = 1^l = 1$$

In the case where $Q \notin \langle P \rangle$, $\{P, Q\}$ forms a basis for $E[n]$ (recall that $E[n]$ has structure $\mathbb{Z}_n \times \mathbb{Z}_n$).

Claim: If $e(P, Q) = 1$, then

$$e(P, [a]P + [b]Q) = 1$$

for all a, b . To show this, we first use linearity to write

$$\begin{aligned}
e(P, [a]P + [b]Q) &= e(P, [a]P)e(P, [b]Q) \\
&= e(P, P)^a e(P, Q)^b \\
&= \zeta_1^a \zeta_2^b
\end{aligned}$$

where $\zeta_1 = 1$ by the alternating property, and $\zeta_2 = 1$ by assumption. This proves the claim, which contradicts non-degeneracy, so it must be that $e(P, Q) \neq 1$ when $Q \notin \langle P \rangle$. \square

Using this condition we can answer the subgroup membership problem with a single pairing computation.

2.5.5 An Easy Instance of the ECDDHP

Verheul [77] uses distortion maps to show that the DDH is efficiently computable for supersingular curves $E : y^2 = x^3 + a$ over \mathbb{F}_{p^2} with order $\#E = p^2 - p + 1$. The reason this particular curve was chosen is because its group of points is isomorphic to the XTR subgroup — a subgroup of \mathbb{F}_{p^6} of order $\#E$. Some cryptosystems are based on XTR [44, 45, 46]. The MOV embedding provides the mapping in one direction from E to the XTR subgroup. The possibility of an inversion of this mapping motivated Verheul to study the DDH in $E(\mathbb{F}_{p^2})$. Although it is a very specific case, the technique is interesting because of the use of pairings. In the process, the important concept of distortion maps were introduced.

Let L be an extension of a field K and let Q be a point in a cyclic subgroup $\langle P \rangle$ of $E(K)$. A *distortion map* $\phi : E(K) \rightarrow E(L)$ maps Q to a point $\phi(Q) \in E(L)$. The result is that Q and $\phi(Q)$ are independent, i.e. $\phi(Q) \notin \langle P \rangle$. A distortion map is often used to replace the trivial pairing $e(P, P)$ with $e(P, \phi(P))$. It was also proven in [77] (Theorem 6) that distortion maps always exist for supersingular curves and can only exist for ordinary curves with embedding degree one.

In this case where $E : y^2 = x^3 + a$, our distortion map will be $\phi : E(\mathbb{F}_{p^2}) \rightarrow E(\mathbb{F}_{p^6})$, For example, if $p \equiv 5 \pmod{6}$, $a \in \mathbb{F}_{p^2}$, $a \notin \mathbb{F}_p$ and is a square but not a cube, we could use

$$\phi(x, y) = (x^p / (\gamma a^{(p-2)/3}), y^p / a^{(p-1)/2})$$

where $\gamma \in \mathbb{F}_{p^6}$ and $\gamma^3 = a$ (this example is from [29], p. 204).

We write the DDH (cf. §2.5.1) problem as: Given P , $X = [x]P$, $Y = [y]P$ and $Z = [z]P$, decide if $Z = [xy]P$

Theorem 5 (Verheul, [77, Th. 3]) *The DDH problem in the group of points of order $p^2 - p + 1$ of any supersingular curve over \mathbb{F}_{p^2} is efficiently computable.*

Proof sketch. First write $\#E$ as $t \cdot v$ where $t = 3^s$ and $\gcd(v, 3) = 1$. Using the Pollig-Hellman algorithm [58], reduce the DDHP in E to the DDHP in the subgroups of order t and v . In the subgroup of order t , the DL is easy, so is the DDH is as well. For the subgroup of order v , we must decide whether $z \equiv x \cdot y \pmod{v}$. We can use

the Weil pairing:

$$e_v(X, \phi(Y)) = e_v(P, \phi(P)^{xy}), \text{ and}$$
$$e_v(P, \phi(Z)) = e_v(P, \phi(P))^z$$

The decision can be made since $z \equiv xy \pmod v$ iff $e_v(X, \phi(Y)) = e_v(P, \phi(Z))$. Each of the above steps are efficiently computable, hence the DDHP can be solved efficiently.

□

Chapter 3

Problem Description

3.1 Parameter Sizes and Security Levels

Implementations of cryptosystems have always been faced with the tradeoff of security and efficiency. The choice of parameter sizes must be made first and foremost with the intention of keeping data secure. This requires an estimate of parameter sizes large enough to prevent cryptanalysis by solving the underlying hard problem (e.g. the DLP). This estimate should include the threat of current and anticipated future attacks. To make the decision more difficult, larger parameters slow encryption and decryption operations. The goal then, is to choose parameters large enough to maintain security, but not so large that efficiency is decreased any more than necessary.

For pairing-based cryptography, the ECDHP, ECDLP and ECBDHP must be difficult in $E(\mathbb{F}_p)$ (we focus on prime fields in this section). Since algorithms for solving the ECDHP and ECBDHP problems compute discrete logarithms, it suffices to ensure the ECDLP is hard. To do this, the subgroup of order n must be large enough to resist square-root attacks such as Pollard's Rho [59] and kangaroo [60] algorithms. The DLP must also be hard in \mathbb{F}_{p^k} , since discrete logs on the curve can be traded for discrete logs in this field via the MOV reduction.

The parameters and implementation decisions are:

1. the extended field size in bits, b_{p^k}
2. the size of the field characteristic
3. the embedding degree k
4. to use an ordinary or supersingular curve
5. the size of the prime order subgroup in bits, b_n

6. the Weil pairing or the Tate pairing

In [43], Lenstra determines the key sizes required by public-key systems to have security comparable to AES (the *Advanced Encryption Standard* or *Rijndael* [26, 57]). The three AES security levels are 128, 192 and 256 bits¹. This was done by estimating the key size required in a public key system to match the security provided at each AES security level. This may be somewhat inaccurate since the security provided by AES is not guaranteed, as it is also estimated. Although it may not provide a concrete recommendation for the size of keys in an implementation, the concept of multiple security levels is too valuable to abandon in a study of efficiency.

Recent work by Kobitz and Menezes [41] adopted the concept of security levels when discussing implementation issues related to pairing-based cryptography. The main benefit is the combination of the implementation decisions 1, and 3 and 5 above. They introduce the ratio

$$\gamma = \frac{\text{bits of } \mathbb{F}_{p^k}}{\text{bits of } n} = \frac{b_{p^k}}{b_n},$$

and give the table:

AES security level	80-bit	128-bit	192-bit	256-bit
Minimum b_n	160	256	384	512
Minimum b_{p^k}	1024	3072	8192	15360
Minimum b_{q^k}	—	4700	12300	24800
$\gamma = b_{p^k}/b_n$	6.4	12	21.3	30

Table 3.1: Minimum bitlengths for n, p^k, q^k . (p is prime, q is a power of 2 or 3. b_{q^k} is the size of \mathbb{F}_{q^k} in bits. Table adopted from [41].)

Table 3.1 gives the minimum bitlengths required to maintain the difficulty of the ECDLP. The 80-bit security level is not one of the AES levels; it was added to include the parameters currently being recommended by some pairing-based literature. Due to the growth of parameter sizes, efficiency will be dramatically affected as the security level increases. This condition is not unique to pairing-based systems; RSA and other public-key systems scale with similar awkwardness [43]. There is still motivation to use pairing-based systems since the features they provide cannot be achieved by

¹AES is a block cipher with a variable key length — it can be 128, 192, 256 bits long. AES is thought to be secure, that is, one can do no better than a brute force search of the keyspace. This is the reason why a larger key gives more security.

other systems. To date, little work has been done to investigate the performance and feasibility of these systems at high security.

The other two decisions are 2 and 4 from our list (on page 28). With respect to the field characteristic, the decision is to either use a small characteristic (2 or 3) or a large characteristic. When \mathbb{F}_q is of small characteristic a more efficient algorithm exists for discrete logarithms, due to Coppersmith [22]. For this reason, the bitlength of q should be much larger than a prime field, as shown in Table 3.1. Many arithmetic and representation enhancements are applicable to fields of small characteristic. This makes them competitive despite their requirement of larger bitlengths.

Large characteristic fields can have a smaller bitlength, however it must still be of size comparable to an RSA modulus providing similar security. Prime fields do not enjoy the arithmetic enhancements of binary and ternary fields. In some cases it is possible to improve the field arithmetic by using GM primes, which we will investigate in Chapter 4. Not unlike the low-characteristic case, a version of the number field sieve has recently been discovered [63], specialized to find discrete logarithms in fields where the characteristic is a GM prime. Once we have quantified the performance of GM arithmetic, we will assess the impact of this discovery.

One reason for debate on whether to use supersingular or ordinary curves is the range of values available for k . As mentioned earlier, supersingular curves have $k \leq 6$. A larger value of k increases security by keeping the DLP in \mathbb{F}_{q^k} difficult, while the cost of curve arithmetic used during pairing computation grows only slowly. With ordinary curves, k is not limited, but constructing curves with a given k can be difficult. A further disadvantage is the lack of distortion maps, which adversely affects efficiency in some applications. Our focus on supersingular curves is motivated by the existence of curve construction methods to allow the use of GM parameters. Comparable methods are not known for non-supersingular curves.

3.2 Tate vs. Weil

The last implementation decision on our list was the choice of pairing. In the work of Boneh and Franklin [16], an example IBE system is provided using the Weil pairing. Their cryptographic scheme was described generally, and requires only a bilinear pairing. As they noted, it can also be implemented with the Tate pairing.

Recall the definitions of the Tate pairing $\langle P, Q \rangle = f_P(D_Q)$, and the Weil pairing $e_n(P, Q) = (-1)^n \frac{f_{n,P}(Q)}{f_{n,Q}(P)}$. In §5.1, we will see how Miller’s algorithm works to compute $f_{n,P}(Q)$, $f_{n,Q}(P)$ and $f_P(D_Q)$. The reason the Tate pairing has been favored is that it requires only a single use of Miller’s algorithm, while the Weil pairing requires two. Since the Tate pairing maps to equivalence classes, the value computed by the pairing is not well defined — but this property is required by most cryptographic applications. Therefore, in addition to the execution of Miller’s algorithm, the Tate pairing requires that the result be raised to the power $(q^k - 1)/n$.

For the lower security levels (80,128-bit c.f. §3.1) considered in the literature, the final exponentiation for the Tate pairing is small enough that it remains the best choice for efficiency. Since the Tate pairing was always favored, a great deal of research has gone into optimizing it, while comparatively little has targeted the Weil pairing. That said, many of the optimizations were “pairing indifferent” since they applied to Miller’s algorithm, which is common to both.

However, if we compare them asymptotically, the Weil pairing requires $2 \log n$ (curve) operations compared to $\log n$ (curve) + $\log(q^k - 1)/n$ (field) operations for Tate. Then we have Weil: $O(\log n)$, Tate: $O(\log q^k)$. Of course this comparison can only be interpreted to mean that *there exist* combinations of n and q^k such that the Weil pairing will provide greater efficiency than the Tate pairing.

The work of Koblitz and Menezes [41] makes this comparison at higher security levels. Their analysis suggests that the crossover point is around the 192-bit security level. We will confirm this experimentally in Chapter 5.

Subsequent to [41], a recent preprint of Granger, Page and Smart [36] describes a method to greatly reduce the work of the Tate pairing’s final exponentiation. The applicability of their technique is limited to ordinary elliptic curves, with embedding degree greater than 6. Using the same cost model as [41] (which we review in Chapter 6) they conclude that the Tate pairing will outperform the Weil pairing in all cases where their method is applicable.

This new work gives only partial resolution to the issue of which pairing to use, as supersingular curves of low embedding degrees may provide efficient implementation (especially $k = 2$) once the benefit and practicality of GM primes is better understood. For this reason, it is still beneficial to know the crossover point in the cost of pairings

at $k = 1, 2(ss)$.[†]

3.3 Quantitative and Practical Issues

The results of the analysis in [41] are, as the authors admit, difficult to compare. Table 3 (of [41], reproduced as Table 6.1) shows the analysis results for $k \in \{1, 2(ss), 2(ns), 6, 12, 24\}$ and security levels 80, 128, 192, 256. The estimates state the cost in multiplications for all options. The difficulty in comparing the analysis arises since the advantage of arithmetic modulo a GM prime is not known. We will study this issue in Chapter 4, and determine the relative performance, by comparing implementations. Our benchmarks will cover a range of sizes and will pay close attention to two practical details largely ignored by previous work. The first is the modular reduction weight, a property of GM numbers which describes the number of operations required to perform a reduction. The other property is whether a GM number is word aligned, which simplifies implementation and yields greater efficiency.

One further parameter choice is intended to speed up computation of Miller’s algorithm. The order of the prime subgroup n , is chosen to have low hamming weight. This reduces the number of add operations performed, since it is quite similar to the “double-and-add” method. A similar outcome is possible by choosing n to be a GM prime, which we discuss in §5.2.

For our study, the only curve construction techniques known to find curves with both n and p generalized Mersenne numbers are for supersingular curves of embedding degree 1 and 2. We will re-examine these two methods, paying special attention to the practical issues, and if possible implement and benchmark the Weil and Tate pairings. These timings will quantify the importance of choosing n as a GM prime. They will also serve to verify the analysis of [41], which suggests that the Weil pairing should be used at higher security levels.

The next two chapters branch out in two directions. Chapter 4 will deal with issues related to the efficient implementation of modular arithmetic. Following it, Chapter 5 will cover Miller’s algorithm in detail, and discuss the issues related to GM primes in the computation of pairings. In Chapter 6 we explain the cost model used by Koblitz and Menezes, and strengthen their analysis. By including our new

[†] “ss” = supersingular; “ns” = nonsupersingular.

knowledge of arithmetic modulo GM primes, we hope to allow comparison across all options. We also discuss the results of Chapters 4 and 5 in the context of pairing-based cryptography and make recommendations for implementation.

Chapter 4

Modular Arithmetic

Part of the cost of computations when our elliptic curve is taken over a finite field is modular arithmetic. Modular (or field) arithmetic is used to implement group operation on the elliptic curve, which is in turn used by Miller's algorithm to compute pairings. Field arithmetic can be divided into two steps; the operation (e.g. multiplication, addition) and reduction. *Modular reduction*, denoted $x \bmod N$, gives the integer remainder when x is divided by N . We will focus on modular reduction.

In this chapter we describe some methods to reduce a number mod N in two cases; for general N , and when N has a special form. We will then describe the experiments performed to compare reduction with generalized Mersenne methods to standard methods. Finally some details of our implementation and the results are presented.

4.1 Modular Reduction by General Moduli

This section reviews some of the methods for modular reduction that do not require the modulus to have a special form.

Classical Reduction

To find the remainder of an integer $x \pmod{N}$, we can use a division. From the quotient-remainder decomposition

$$x = qN + r \quad 0 \leq r < N .$$

We can solve for r and replace q with $\lfloor x/N \rfloor$ to arrive at

$$r = x \bmod N = x - N \lfloor x/N \rfloor .$$

As one would expect, this approach is only as fast as the division step, and the method is not competitive for large inputs. Classic division algorithms are discussed in [25] and [39].

Barrett Reduction

Barrett reduction computes $x \pmod{N}$ by estimating $\lfloor x/N \rfloor$ using cheap operations. Description of the algorithm is best left to references as we will not be using it in this work. Interested readers can find a description in the original paper [15] and good presentations in the books [50, 38, 25].

Montgomery Reduction

Montgomery reduction was first described in 1985 by P. Montgomery [54]. The Montgomery and Barrett methods have the same asymptotic complexity [25] and have similar practical performance [25, 17, 73]. To compute $x \pmod{N}$, the idea is to compute $xR^{-1} \pmod{N}$ for a carefully chosen R .

We repeat Theorem 9.2.1 as stated in [25]. This theorem shows how to compute $xR^{-1} \pmod{N}$.

Theorem 6 (*Montgomery*) *For coprime integers N, R , let $N' = -N^{-1} \pmod{R}$. For any integer x , the number*

$$y = x + N(xN' \pmod{R})$$

is divisible by R , with

$$y/R \equiv xR^{-1} \pmod{N}$$

Further, if $0 \leq x < RN$, the difference $y/R - (xR^{-1} \pmod{N})$ is either 0 or N .

The operations of Theorem 6 are quite simple when R is chosen to be $2^s > N$. Computing $y/R \equiv xR^{-1} \pmod{N}$ can be effected with two multiplications (of numbers roughly the size of N).

Since $R^{-1}\mathbb{Z}_N$ results in a complete set of residues mod N we can compute $xR^{-1} \pmod{N}$ then recover $x \pmod{N}$ by a multiplication by R . This is most useful when many multiplications are required (e.g. exponentiation), they can all be done in $R^{-1}\mathbb{Z}_N$ and only the result is multiplied by R . In this way, the cost of changing the representation is spread across many operations.

4.2 Modular Reduction by Special Moduli

When N has special structure, and we wish to compute $x \pmod{N}$, taking advantage of this structure can lead to faster algorithms. This section presents some of them.

Modular Reduction by Mersenne Numbers

It is well known that the Mersenne numbers $N = 2^k - 1$ allow fast modular reduction. Instead of doing the reduction using costly integer division, or even multiplications, it can be done using only addition mod N . If x is a $2k$ -bit integer less than N^2 , x can be written as $x = T \cdot 2^k + U$. In binary T is the k most significant bits and U is the k least significant bits. Reduction is effected by the addition:

$$x \equiv T + U \pmod{N}$$

which we will explain using (4.1) below. On a machine with a 32-bit word size, the case $k = 32$ is especially simple, requiring an addition mod N . Addition mod N is a simple addition followed by a comparison and possibly a subtraction. Further details are available in [25, 24].

Modular Reduction by Crandall Numbers

The scheme of the previous section is excellent when one needs arithmetic modulo a Mersenne number, but is unfortunately limited to this special class of numbers. In applications requiring a prime modulus, the choice is limited to the 43 known Mersenne primes.

Crandall [25] offers a partial generalization, to *pseudo-Mersenne numbers* $2^k + c$ where $|c|$ is small enough to fit in a machine word. Sometimes, the condition $\log_2 c < \frac{1}{2}k$ is used in the definition [8]. Algorithm 9.2.13 for reduction modulo a pseudo-Mersenne number N in [25] is based on the theorem

$$x \equiv (x \bmod 2^k) - c \lfloor x/2^k \rfloor \pmod{N} \quad (4.1)$$

The algorithm uses only shifts and subtractions, and multiplication by c . Due to patent issues [23], it is avoided in cryptographic applications.

It may also be generalized to the Proth numbers of the form

$$N = q \cdot 2^k + c$$

when q and c fit in a single word [25].

Equation (4.1) is especially simple in the Mersenne case when $c = 1$. The lower bits are equivalent to reduction mod 2^k , which is “close to” the correct answer. The upper bits are added as a sort of correction factor, to make up for the fact that we were supposed to reduce mod $2^k - 1$. When $|c|$ is larger than 1, we must add this “correction factor” for every $|c|$, giving (4.1).

4.3 Generalized Mersenne Numbers

The work described in this section is due to Jerome Solinas [74]. Shortly after, field arithmetic modulo specific generalized Mersenne numbers (or *GM numbers* for short) was recommended by the NIST for use in elliptic curve cryptosystems by the US federal government.¹

The generalization presented by Solinas uses numbers of the form

$$2^d - c_1 2^{d-1} - \dots - c_d$$

for integers c_i . Generally c_i is chosen to be in $\{1, -1, 0\}$ or integers with a small absolute value, however the method could work for any $c_i \in \mathbb{Z}$. Throughout this section, we will use the class of numbers

$$p = 2^{3k} - 2^k + 1 \tag{4.2}$$

as an example (as in [74]), and use $k = 32$ to show concrete examples. We will also assume a wordsize of 32 bits. For $k = 32$,

$$p = 2^{3 \cdot 32} - 2^{32} + 1 = 79228162514264337589248983041.$$

p is a 96 bit prime and can be represented as three 32-bit words. The integers mod p can therefore be represented with 3 words or less.

When working in \mathbb{Z}_p , if we reduce after each multiplication we will never have to reduce a number larger than $(p - 1)(p - 1) < p^2$. Let n be an integer less than p^2 . Then n has $2 \cdot 3k$ bits (six words in our example). We can write

$$n = \sum_{j=0}^5 A_j \cdot 2^{jk}.$$

¹The National Institute for Standards and Technology made this recommendation in [56], *Appendix 6: Recommended elliptic curves for federal government use*.

For our example, this would look like

$$\begin{array}{rcccccc}
 & & & & & & A_0 \\
 + & & & & & A_1 & 0 \\
 + & & & & A_2 & 0 & 0 \\
 + & & & A_3 & 0 & 0 & 0 \\
 + & & A_4 & 0 & 0 & 0 & 0 \\
 + & A_5 & 0 & 0 & 0 & 0 & 0 \\
 \hline
 = n = & w_5 & w_4 & w_3 & w_2 & w_1 & w_0
 \end{array}$$

So if the large number n is represented as k -bit words (w_0, \dots, w_{k-1}) , the $A_j = w_j$. Note that it is equally simple for larger k which are divisible by the word size. When k is not a multiple of the word size, working with the A_j becomes more complicated, requiring bit shifts.

We now search for some B_i that are linear combinations of the A_j such that $n \pmod{p}$ is congruent to their sum (and difference). It is similar in spirit to the Mersenne case, we will reduce mod 2^k (take the lower bits) and apply some “correction factor” by adding selected upper words.

Computing this sum will then give $n \pmod{p}$ using only additions (and subtractions). Continuing with our example, since we are reducing mod p we only need $3k$ bits (or 3 words if $k = 32$).

$$n = \sum_{j=0}^5 A_j \cdot 2^{jk} \equiv \sum_{i=0}^2 B_i \cdot 2^{ik} \quad (4.3)$$

In our example, this sum looks like

$$\begin{array}{rcccc}
 & & & & B_0 \\
 + & & & & B_1 & 0 \\
 + & & & B_2 & 0 & 0 \\
 \hline
 = n \pmod{p} = & u_2 & u_1 & u_0
 \end{array}$$

which gives us the words u_i of $n \pmod{p}$. If we revisit the Mersenne case $m = 2^k - 1$ where k is the wordsize we have $n \pmod{m} = B_0 = A_1 + A_0$, which amounts to the lower bits of n plus the upper bits.

The work comes when we want to find the B_i for arbitrary GM primes, and Solinas presents an excellent method. First, think of p as a polynomial in 2^k . Then (4.2)

becomes:

$$p = f(2^k), \quad f(t) = t^3 - t + 1.$$

The general form of this polynomial is written

$$f(t) = t^d - c_1 t^{d-1} - \dots - c_d \quad (4.4)$$

From this we can see that in general there will be $2d$ As and d Bs (which generalizes the sums of (4.3)).

Then write n as

$$n = \underbrace{(A_0 \dots A_{d-1})}_{\text{low words}} \begin{pmatrix} 1 \\ \vdots \\ t^{d-1} \end{pmatrix} + \underbrace{(A_d \dots A_{2d-1})}_{\text{high words}} \begin{pmatrix} t^d \\ \vdots \\ t^{2d-1} \end{pmatrix}. \quad (4.5)$$

We now compute $t^d \bmod f(t), \dots, t^{2d-1} \bmod f(t)$ (and write in matrix form):

$$\begin{pmatrix} t^d \\ \vdots \\ t^{2d-1} \end{pmatrix} \equiv X \begin{pmatrix} 1 \\ \vdots \\ t^{d-1} \end{pmatrix} \pmod{f(t)}$$

for a d by d matrix X . Then substitute this into (4.5)

$$n \equiv (A_0 \dots A_{d-1}) + (A_d \dots A_{2d-1}) \cdot X \begin{pmatrix} 1 \\ \vdots \\ t^{d-1} \end{pmatrix} \pmod{f(t)}$$

to get a rule for reduction \pmod{p} .

The square matrix X of size d :

$$X = \begin{bmatrix} X_{0,0} & \dots & X_{0,d-1} \\ \vdots & \ddots & \vdots \\ X_{d-1,0} & \dots & X_{d-1,d-1} \end{bmatrix}$$

can also be created recursively by the following formulas. The first row is the coefficients in reverse:

$$X_{0,j} = c_{d-j} \quad \text{for } 0 \leq j < d$$

while subsequent rows are given by

$$X_{i,j} = \begin{cases} X_{i-1,j-1} + X_{i-1,d-1}c_{d-j} & \text{for } j > 0 \\ X_{i-1,d-1}c_d & \text{for } j = 0 \end{cases}.$$

This somewhat cryptic construction is derived from creating a linear feedback shift register (LFSR) over \mathbb{Z} based on $f(t)$ with initial fill $000\dots 1$.

The B_i are given by the matrix equation:

$$[B_0 \quad \dots \quad B_{d-1}] = [A_0 \quad \dots \quad A_{d-1}] + [A_d \quad \dots \quad A_{2d-1}]X. \quad (4.6)$$

Back in our example, where $f(t) = t^3 - t + 1$, $d = 3$, $c_1 = 0$, $c_2 = 1$, $c_3 = -1$, we create the matrix:

$$X = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ -1 & 1 & -1 \end{bmatrix}$$

which we substitute into equation (4.6)

$$[B_0 \quad B_1 \quad B_2] = [A_0 \quad A_1 \quad A_2] + [A_3 \quad A_4 \quad A_5] \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ -1 & 1 & -1 \end{bmatrix}$$

to get the B_i in terms of the A_j

$$\begin{aligned} B_0 &= A_0 - A_3 - A_5 \\ B_1 &= A_1 + A_3 - A_4 + A_5 \\ B_2 &= A_2 + A_4 - A_5. \end{aligned}$$

Now substitute into the sum $n \equiv \sum_{i=0}^2 B_i \cdot 2^{jk}$ from (4.3) to get:

$$\begin{aligned} n &\equiv B_0 + 2^k B_1 + 2^{2k} B_2 \\ &= A_0 - A_3 - A_5 + 2^k A_1 + 2^k A_3 - 2^k A_4 + 2^k A_5 + 2^{2k} A_2 + 2^{2k} A_4 - 2^{2k} A_5 \end{aligned}$$

With a little care, this can be reorganized into contiguous pieces (words) of n (MSB is leftmost).

$$\begin{aligned} T &= A_2 \cdot 2^{2k} + A_1 \cdot 2^k + A_0 \\ S_1 &= A_4 \cdot 2^{2k} + A_3 \cdot 2^k \\ S_2 &= A_5 \cdot 2^k \\ D_1 &= A_5 \cdot 2^{2k} + A_4 \cdot 2^k + A_3 \\ D_2 &= A_5 \end{aligned}$$

The result is the simplified mod operation

$$n \pmod p \equiv T + S_1 + S_2 - D_1 - D_2$$

which requires only two additions and two subtractions mod p .

The Modular Reduction Weight

The above example required 2 additions and 2 subtractions mod p . We say the *modular reduction weight* (or MRW) of the polynomial $f(t) = t^3 - t + 1$ is 4, and write $\text{MRW}(f) = 4$. How can the weight be computed in general?

Using the matrix X we sum the positive entries in column j

$$Y_j = \sum_{\substack{i \\ X_{i,j} > 0}} X_{i,j}$$

then define Y_{\max} as the largest Y_j for $0 < j < d - 1$.

Similarly we sum the absolute values of negative entries in column j

$$Z_j = \sum_{\substack{i \\ X_{i,j} < 0}} |X_{i,j}|$$

and defined Z_{\max} as the largest in X .

The total number of modular reductions required to reduce $n \bmod p = f(2^k)$ is:

$$\text{MRW}(f) = Z_{\max} + Y_{\max} .$$

Since a large MRW can slow the speed of modular reduction to the point where other methods become more efficient, we will search for polynomials of low MRW.

Finding Low Weight Polynomials

In this section, we review some criteria that can be used to find polynomials of low modular reduction weight (also due to Solinas [74]).

An integral polynomial $f(t)$ is called *reduced* if for each $l \in \{2, \dots, d\}$ we have $c_j \neq 0$ for some $j \not\equiv 0 \pmod{l}$. Let's consider an example.

$$f(t) = t^4 - c_1 t^3 - c_2 t^2 - c_3 t - c_4$$

Suppose that f is reduced. For $l = 2$, c_1 or c_3 must be nonzero, when $l = 3$, one of $c_{1,2,4}$ is nonzero, and for $l = 4$ some $c_{1,2,3}$ is nonzero. Using these constraints we can construct some reduced polynomials, say $g(t) = t^4 - 2t - 7$, or $h(t) = t^4 + t^3 - t^2 + 3$.

Proposition 10 of [74] proves that no generality is lost by considering only reduced polynomials. If a number n is represented $n = f(2^k)$ for a polynomial $f(t) \in \mathbb{Z}[t]$,

it can also be represented as $n = g(2^{k'})$ where g is a reduced polynomial. Further, $\text{MRW}(f) = \text{MRW}(g)$ and if k is a multiple of the word size, then so is k' . This allows a search for low weight polynomials to only consider reduced polynomials. Proposition 11 states: “if $f(t) = tg(t)$ then $\text{MRW}(f) \geq \text{MRW}(g)$ ”. The implication of this result is that candidates in a search for low-weight polynomials should have the constant term $c_d \neq 0$.

The polynomial in (4.4) is said to be *positive* when all $c_j > 0$. Note that this is slightly misleading since c_j appears as $-c_j$ for all j in equation (4.4). A positive polynomial actually has *negative* coefficients. A new concept is defined, the *modular reduction complement* of a polynomial $f(t)$,

$$f^*(t) = t^d - \sum_{j=1}^d (-1)^j c_j t^{d-j}$$

and it is shown that $\text{MRW}(f(t)) \leq \text{MRW}(f^*(t))$. The implication for searching in this case is that only positive polynomials need be checked. Once those with minimum weight are identified, the weights of their complements can be checked as well.

Search Strategy

Collecting up these restrictions, we have significantly pruned the number candidate polynomials that can have low reduction weight.

- Reduced polynomials
- Polynomials with constant terms: $c_d \neq 0$.
- Positive polynomials: $c_j > 0$ for all j .

If the goal is to represent prime moduli with the polynomial, there are two further conditions

- f must be irreducible, otherwise it will admit an algebraic factorization, making $f(2^k)$ composite.
- The constant term should be odd, otherwise $f(m)$ will be even.

4.3.1 Related Work

Generalizing Further

In a similar fashion to Solinas, Chung and Hasan [21] also generalize Mersenne numbers. They take the polynomials $f(t)$ of the previous section, and allow t to be any integer, not only a power of two. This gives a much larger choice of moduli, and decreases the specificity of an implementation. A single implementation could handle multiple polynomials $f(t)$ for any value of t . Unlike Solinas, their work presents a modular multiplication algorithm and not a reduction algorithm. Their method represents multiplicands as polynomials in $\mathbb{Z}[t]$, then computes their product in $\mathbb{Z}[t]/f(t)$ to get the result mod $f(t)$. For modular multiplication, their method essentially reduces computation by $9n^2$ multiplications but increases it by $3n$ divisions (where n is the word length of t). They also give some criteria for selecting polynomials yielding the best performance.

Subsequently, the work of Bajard [9] approaches the problem by representing numbers mod N in any base β , and performing polynomial arithmetic in $\mathbb{Z}[\beta]$. Their operation counts show their algorithms using n^2 fewer multiplications than Montgomery, trading them for approximately $6n^2$ additions. The authors state that the incentive to use this representation is the ability to use moduli which are not possible with the methods of Solinas or Chung and Hasan.

In practice, it seems likely that these methods are faster than Montgomery modular multiplication. Comparison is difficult however, and without practical tests it is difficult to gauge the degree of the improvement.

Optimal Extension Fields

Although *optimal extension fields* (OEF) will not be a part of this study, we will briefly describe them since they provide fast implementations for cryptographic applications. Optimal extension fields were first proposed by Baily and Paar [8, 7]. An OEF is created using an irreducible polynomial as we described in section 2.1.2. The field \mathbb{F}_q , where $q = p^m$ is represented

$$\mathbb{F}_q = \frac{\mathbb{F}_p[x]}{\langle f(x) \rangle}$$

with the following two properties:

1. The prime $p = 2^k + c$ is a Crandall number (described in section 4.2) which is less than, but close to the word size of the processor, and
2. $f(x) = x^m - b$.

This allows fast subfield arithmetic, since elements of \mathbb{F}_p are single word integers, which can be multiplied with fast modular reduction. The second property allows fast modular reduction in the extension field. A key result of [8] states that a polynomial in $\mathbb{F}_q[x]$ can be reduced modulo $x^m - b$ using only $(m - 1)$ multiplications by b and $(m - 1)$ additions. When $b = 2$, the multiplications can be traded for shifts.

Performance Studies

In this section we review the results of some studies comparing the performance of modular reduction/multiplication methods discussed in the above section. Due to the popularity of elliptic curve cryptography, many such studies have been done with binary fields (see the references of [38, 18]). Fewer recent studies exist which focus on the performance of operations in \mathbb{F}_p .

The 1993 paper of Bosselaers, Govaerts and Vandewalle [17], compares classical, Montgomery and Barrett modular reduction functions. Unfortunately, the work of Solinas had not been published at this time. They compared the time required to reduce a number and the time for modular exponentiation. Their C implementations on an 80386 PC showed similar performance of all three methods. They found Barrett slightly faster than classical, and Montgomery slightly faster than Barrett for both reduction and exponentiation. The length of the moduli used in this study ranged from 128 to 1024 bits.² Some later work [78] found close similarity between the Barrett and Montgomery reduction methods for a 192-bit modulus. Five variations of the Montgomery algorithm were compared in [42], to examine the subtle differences in their performance.

Crandall and Pommerance [25] give the following comment on the reports that cryptographic applications see slightly better performance with the Montgomery method: “reaching the asymptotically best complexity for the Montgomery method is easier than for the Barrett method.”

²At the time these were considered large, but times have changed.

A relevant and more recent comparison of field arithmetic is provided by Smart [73]. He compares timings for operations in the following four fields:

- Prime fields: \mathbb{F}_p for a general prime, and a generalized Mersenne prime
- Extension fields: binary fields \mathbb{F}_{2^m} , and optimal extension fields \mathbb{F}_{p^m}

Reduction in the general prime field used Montgomery's algorithm. For the prime fields, $p = f(2^{64}) = 2^{192} - 2^{64} - 1$. Here, f is the polynomial we used for our examples in section 4.3. It has modular reduction weight four, and was the only prime used in the comparison. In all timings presented, the OEF implementation was fastest.

The most relevant comparison for our work was between the prime field implementations. With these parameters, multiplication using a GM prime provided a 25% speedup on the SPARC architecture. On the Pentium CPU, the prime fields require exactly the same amount of time for modular multiplication. The author offers no explanation for this difference. We are left with the impression that the author's main goal was to verify the efficiency of using an OEF. It is difficult to draw general conclusions since these timings are specific to prime fields of order roughly 192-bits. It also provides no guidance to judge the effect of the MRW.

To our knowledge these performance studies are the best available. This leaves us without precise measurements of the time required for Solinas arithmetic. The rest of this chapter will establish some.

4.4 Internal Storage of Large Integers

The implementations described in this section will use the GNU Multiple Precision Library (GMP) [2]. It is a portable C library for arbitrary precision arithmetic on integers, rational numbers, and floating-point numbers. GMP is the fastest such library available in the public domain, and is used in many applications. It aims to provide the fastest possible arithmetic for all applications that need higher precision than is provided by the basic C types.

The GMP library stores large numbers with a signed magnitude representation. This representation is both simple and intuitive as it most closely resembles hand written notation [39, 37]. A big number is represented as an array of unsigned integers, called the *limbs*. This is the magnitude of the number. A signed integer keeps track

of the number of limbs and the sign of the number. A final variable in the structure records the space allocated for the array. In our benchmarks, we allocate the space required for our operands when they are declared, avoiding slow reallocations. This is possible since the size of the operands is bounded by N^2 when working mod N provided we reduce after every operation.

GMP provides two APIs for development. The first is a high level API which manages memory allocated for operands and the result and keeps track of the size of the result. It requires the least effort from the user, and provides a consistent interface.

The second API is used to implement the first. These low-level functions are also made available for time-critical user code. It requires that the caller specify which limbs of the operands should be used. The caller must also ensure that sufficient space is allocated and update the size variable.

4.5 Limb Modular Reduction Weight

The modular reduction weight used by Solinas in [74] measures the total number of operations required to perform a modular reduction. When comparing moduli of varying MRW, we found that moduli of different MRW could have nearly the same practical performance. Figure 4.1 shows plots of the time required to perform 10^6 reductions modulo numbers of MRW 2,3,4,5. (Note that this is the time to compute only a modular reduction, and *not* a modular multiplication.) The time in seconds is the average of 5 runs — which had very close timings, the largest standard deviation observed was 0.000122.

One would expect the computation time to increase regularly with the MRW. However, we observe a large jump between 2 and 3, then 4 and 5 are nearly the same with 5 slightly outperforming 4. Although this difference was not crucial, as will be shown, the issue was puzzling prompting further investigation.

This can be explained by considering a more precise measure, which we call the *limb modular reduction weight* (LMRW). It refines the MRW by counting the number of limbs which will be added/subtracted during the reduction. Unlike the MRW, which ignores the sizes of the operands involved, the LMRW is closer to the number of machine operations (provided $k \equiv 0 \pmod{w}$, where w is the word size, cf. §4.3).

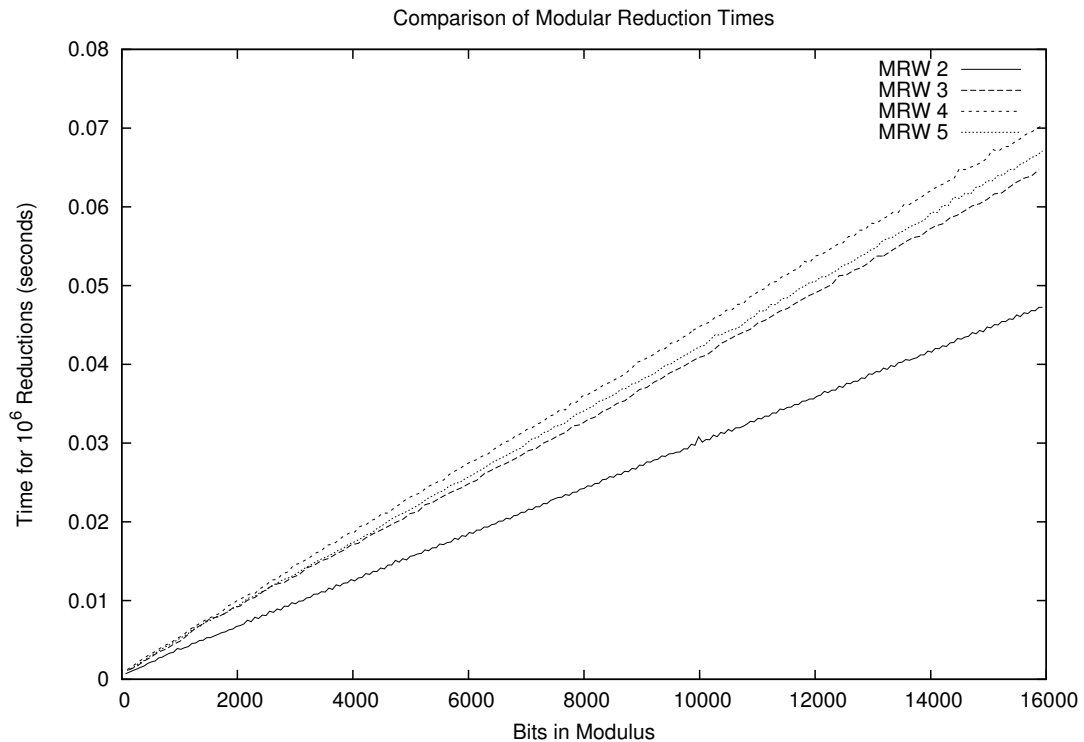


Figure 4.1: Time required to compute $m \pmod{N}$ for 10^6 integers $N < m < N^2$.

The LMRW is computed as follows:

$$\text{LMRW}(f) = d + \sum_{i=0}^d \sum_{j=0}^d |X_{i,j}|$$

where $d = \deg(f)$ and X is the matrix from section 4.3. This counts the number of terms A_j in

$$\sum_{j=0}^d B_j = A_0 \pm \dots + A_1 \pm \dots + A_d \pm \dots$$

The timings in Figure 4.1 are more logical knowing the polynomials of MRW 2,3,4,5 have LMRW 5,13,10,10. Although closer to the observed data, the LMRW is still not perfect since there are two factors it does not measure. The first is the overhead of the function call for each operation counted by the MRW. The second is a quirk of the GMP library — addition of operands having the same number of limbs is faster than addition of operands of differing size. ³

³There is no magic here, the library simply has an assembly implementation of the `mpn_add_n()` function, and a C implementation of the `mpn_add()` function. In an effort to simplify comparison we used only `mpn_add()`, however the source code reveals that it will make a call to the faster function if the operands have the same size. These issues are discussed in [37], Section 8: *Low-level Functions*.

The bound $\text{LMRW} \leq (d + 1)\text{MRW}$ suggests that when searching for low weight polynomials of “reasonably” small degree it is sufficient to compute only the MRW. Since the inaccuracy of the MRW increases with d , for larger d the LMRW might also be considered. In what follows, where our interest is modular multiplication, the time for reduction is dwarfed by the time of a multiplication. We found the MRW sufficient.

4.6 A Family of Generalized Mersenne Numbers

Many of the GM numbers in Table 4.2 are of the form

$$t^d - t^{d-1} + 1 .$$

This family has a predictable MRW of $2d - 1$, and

$$\text{LMRW} = \frac{d(d-1)}{2} + 2(d-1) + d$$

which are obvious when one considers the matrix X which always takes the form

$$X = \begin{bmatrix} -1 & 0 & 0 & 0 & \dots & 0 & 1 \\ -1 & -1 & 0 & 0 & \dots & 0 & 1 \\ -1 & -1 & -1 & 0 & \dots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & & \vdots & \vdots \\ -1 & -1 & -1 & -1 & \dots & -1 & 1 \\ -1 & -1 & -1 & -1 & \dots & -1 & 0 \end{bmatrix}$$

Knowledge of such families removes the need for polynomial searches and computation of the MRW, we can simply test the primality of $f(2^k)$ at the desired bitlevel. Table 4.1 shows a list of primes of this form. Unfortunately, they are not abundant. Since the coefficients of X are always 1, -1, 0, this family of GM numbers would allow simpler code generating software. We present this family as an example to supplement those presented by Solinas [74], and leave it as an open problem to find others. As well, are there any other properties of X which lead to desirable implementations, or shorter searches for primes?

Bitlength	p	MRW
3680	$2^{5 \cdot 32 \cdot 23} - 2^{4 \cdot 32 \cdot 23} + 1$	9
576	$2^{9 \cdot 32 \cdot 2} - 2^{8 \cdot 32 \cdot 2} + 1$	17
320	$2^{10 \cdot 32} - 2^{9 \cdot 32} + 1$	19
4352	$2^{136 \cdot 32} - 2^{135 \cdot 32} + 1$	271

Table 4.1: Table of all primes of the form $p = f(2^{32 \cdot k})$ for $f(t) = t^d - t^{d-1} + 1$, such that $\text{bitlength}(p) < 16000$ and $d < 10^4$.

4.7 Modular Arithmetic Experiments

The purpose of this experiment is to compare the modular multiplication methods described in Section 4. For comparison, we will use two baselines. The first will be the standard GMP library functions for multiplication and reduction.

The second baseline will be Montgomery multiplication, the REDC algorithm. We will use the implementation from the GMP-ECM project. (The GMP-ECM project implements the elliptic curve method for factoring large integers [1].) It is optimized and implemented at the same level as our generalized Mersenne implementations. Their REDC implementation consists of a GMP multiplication followed by a special reduction routine.

The algorithm for modular reduction used in the GMP-ECM implementation is one of the following three.

1. Montgomery's REDC algorithm [54] at the word level (REDC1). It is quite fast for small numbers, but has quadratic asymptotic complexity.
2. GMP arithmetic [37]. Has some overhead when compared to (1) for small sizes, but is faster than (1) for larger sizes since it has quasi-linear asymptotic complexity.
3. Montgomery's REDC algorithm at high level (REDC2). This essentially replaces each division by two multiplications. Slower than (1) and (2) for small inputs, but better for large or very large inputs.

(Source: gmp-ecm README file).

The choice of algorithm depends on the size of the modulus, with appropriate thresholds determined for a given machine by a tuning program. The program multiplies progressively larger integers using all three algorithms until the crossover points

are identified. After running the tuning program and some initial timings, we configured the library to use REDC1 for operands of size less than 111, GMP arithmetic for operands with 112 to 204 limbs, and REDC2 for operands with 205 limbs or greater.

Since all of our modular multiplication implementations use the same multiplication code, this makes for a level comparison. As well, the REDC algorithms are implemented using the low-level API of GMP, like the other methods to be compared. The only difference is the reduction function used.

For the polynomials used in the GM reductions, we use a variety of MRW, to better understand the impact of this weight on performance. We would like to get a sense of what is a “high” weight, and at what point performance is significantly affected.

The timings consist of the time required for 10^4 modular multiplications, with operands chosen as follows. Choose 10^4 numbers m evenly spaced between 1 and n , then compute $m(m - 1)$ using the GMP multiply function, then reduce using one of the mod functions. We favor this choice of operands over a random selection for two reasons. It allows our experiments to be more easily reproduced, and ensures that all mod functions work on identical operands. Second, we are guaranteed to test the functions on a wide variety of input sizes.

Our test system runs Gentoo Linux, with a 2.6 kernel in single user mode. The CPU is an Intel Pentium 4 at 3 GHZ. Complete system details including compiler version and flags are included in Appendix B.

In this implementation, we will not differentiate between prime and composite moduli. The methods work the in same way; the character of the modulus has no impact on efficiency. The baselines (which do not exploit the structure of the modulus) were run at various bitlevels with the modulus $2^k - 1$.

We have implemented multiple mod operations, for families of numbers $f(2^k)$ given by the polynomials f in Table 4.2.

Automatic Generation of Reduction Routines

Fortunately, the method of Solinas described in Section 4.3 lends itself well to automation. To allow timely implementation of many reduction routines we created a software tool to generate a GMP implementation for reduction modulo $f(2^k)$ where

$f(t)$	MRW	LMRW
$t - 1$	1	2
$t^2 + t + 1$	2	5
$t^4 - t - 1$	3	13
$t^3 - t + 1$	4	10
$t^3 - t^2 + 1$	5	10
$t^5 - t^3 + 1$	6	19
$t^4 - t^3 + 1$	7	16
$t^3 - t^1 + 3$	8	18
$t^5 - t^4 + 1$	9	23
$t^8 - t^7 + 1$	15	50
$t^{12} - t^{11} + 1$	23	100
$t^{15} - t^{14} + 1$	29	148
$t^{18} - t^{17} + 1$	35	205
$t^{31} - t^{31} + 1$	63	590
$t^{16} - t^{10} + t^7 + t^3 + 17$	119	798
$t^{71} - t^{58} - t^{56} - t^{32} + t^{19} + t^{17} - 1$	188	2967

Table 4.2: Polynomials with a variety of modular reduction weights used. The moduli for benchmarking were set to $f(2^k)$ for all $k \equiv 0 \pmod{32}$ such that $\log_2 f(2^k) < 16000$.

$k \equiv 0 \pmod{32}$ and f is a monic polynomial in $\mathbb{Z}[t]$. Although the polynomials used in our experiments and discussed in related work tend to have coefficients in $\{1, -1, 0\}$ this does not exclude other monic polynomials in $\mathbb{Z}[t]$. In general, due to the way X is constructed higher coefficients lead to higher MRW.

In practical settings, such a tool would be especially important due to the specificity of a routine to a specific polynomial. An application requiring a change of modulus would likely require a change of polynomials — which would force the implementation of a new reduction routine. Although implementation “by hand” may be reasonable at lower modular reduction weights, it is tedious and error-prone.

4.8 Results

We first present an overview of the results with Figure 4.2, which shows performance of many MRW across varying bitlevels. The times for MRW 1-35 were so close that their plots appear as a thick line on the chart. This plot also includes the time required for the GMP multiplication, to show the fraction of the overall time used by the multiplication.

The following figure — Figure 4.3, gives a clearer view, showing only selected modular reduction weights. The precise numbers at key bitlevels are presented in tables 4.3, 4.4, 4.5 and 4.6 below. The telling column in these tables is “Fraction”; which measures the time of the GM method as a fraction of the REDC baseline. The “Security Level” column represents the security level which a field of this size would provide to paring-based cryptographic applications (as defined in §3.1).

We note a few things from these charts. In all cases, the performance gained from using a GM modulus becomes larger as the security level increases. This was mainly due to the quadratic behavior of the baseline. At very low bitlevels (say less than 1500), we find very little improvement, and depending on the MRW, can even give *worse* performance than standard methods. In Figure 4.4, we zoom in on Figure 4.3 to illustrate performance at bitlevels less than 1500.

We also notice that performance differs only slightly between the different MRW implemented and timed. Given the data presented, we conclude that a modular reduction weight of less than 200 will give increased performance, provided the operands are large enough.⁴ Of course larger MRW can still be beneficial at higher bitlevels, but these cases must be considered on an individual basis.

This increases the flexibility when choosing moduli in curve construction techniques. It can also be used to speed searches for prime moduli of practical value, if a polynomial $f(t)$ has a large MRW it can be discarded immediately (i.e. before the primality tests of $f(2^{32}), f(2^{64}), \dots$). The only drawback to large weight reduction functions is the corresponding increase in implementation complexity. Implementation by hand becomes impractical, and a robust method of automatically generating the mod functions becomes essential. The size of the code increases with the MRW, which may be a concern for some implementations.

A final note about the timings presented in this chapter is that they are all the time in seconds to compute 10^4 modular multiplications, where the operands are evenly distributed in \mathbb{F}_p . Dividing the time presented by 10^4 gives the average time in seconds for a single modular multiplication. Again, the times presented in the tables and plots are the average of five runs, the largest standard deviation observed was 0.00474.

⁴See Figures 4.2, 4.3 and 4.4.

Security Level	Bits	REDC	MRW 1	Fraction
80	1024	0.0955	0.0471	0.5005
128	3072	0.7190	0.2803	0.3898
192	8192	3.9788	1.4283	0.3589
256	15360	10.4551	3.6248	0.3467

Table 4.3: Comparison of REDC baseline to MRW 1.

Security Level	Bits	REDC	MRW 35	Fraction
80	1152	0.1192	0.0922	0.7734
128	3456	0.9194	0.4370	0.4753
192	8640	4.4006	1.7849	0.4056
256	15552	10.7895	4.0909	0.3791

Table 4.4: Comparison of REDC baseline to MRW 35.

Security Level	Bits	REDC	MRW 63	Fraction
80	1024	0.0955	0.1028	1.0764
128	3072	0.7190	0.4093	0.5692
192	8192	3.9788	1.7574	0.4416
256	15360	10.4551	4.2408	0.4056

Table 4.5: Comparison of REDC baseline to MRW 63.

Security Level	Bits	REDC	MRW 188	Fraction
80 – 128	2272	0.4011	0.5116	1.2754
128 – 192	4544	1.4318	1.0540	0.7361
\approx 192	9088	4.6656	2.5227	0.5407
\approx 256	15904	11.4698	5.4706	0.4769

Table 4.6: Comparison of REDC baseline to MRW 188. “Security Level” displays a range or approximate level when the bitsize of $f(2^k)$ differs significantly from the bitsize required by the security level.

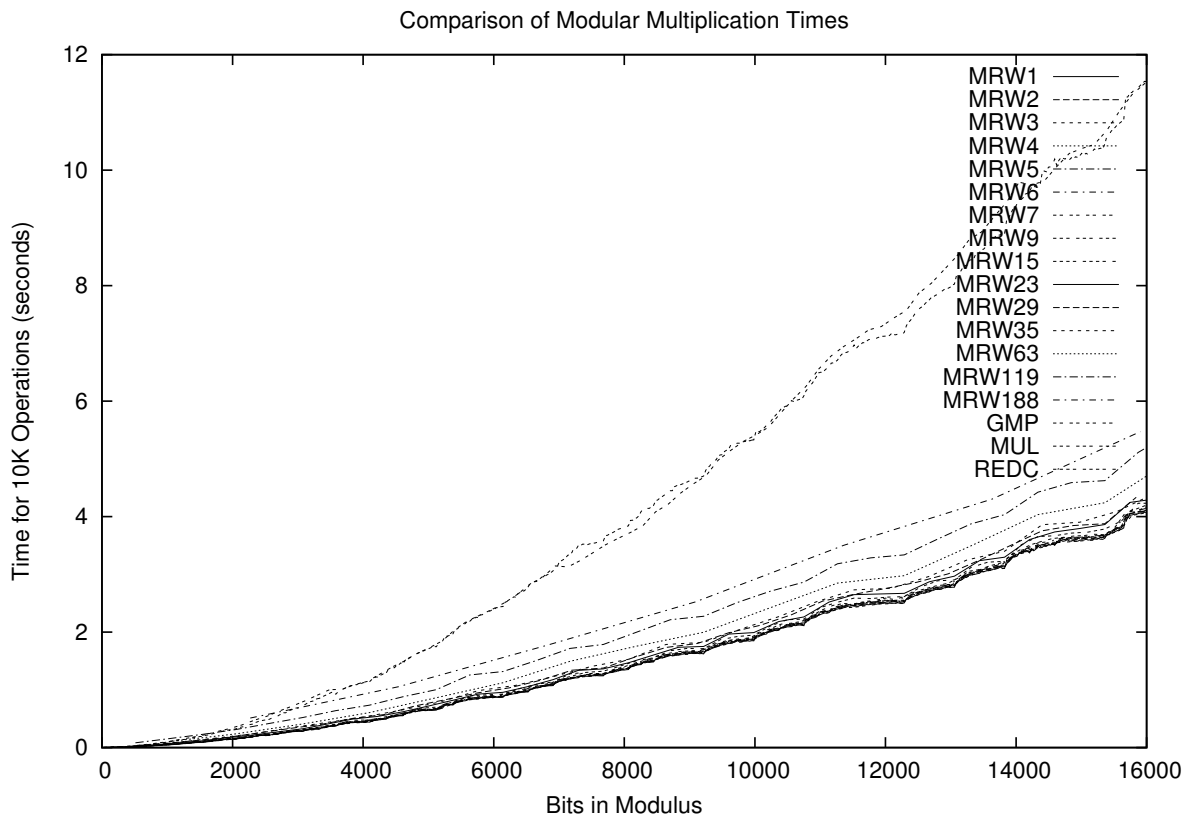


Figure 4.2: Modular multiplication times for varying MRW, the REDC baseline and the standard GMP library implementations. Most of the MRW plots coincide to form the thick line.

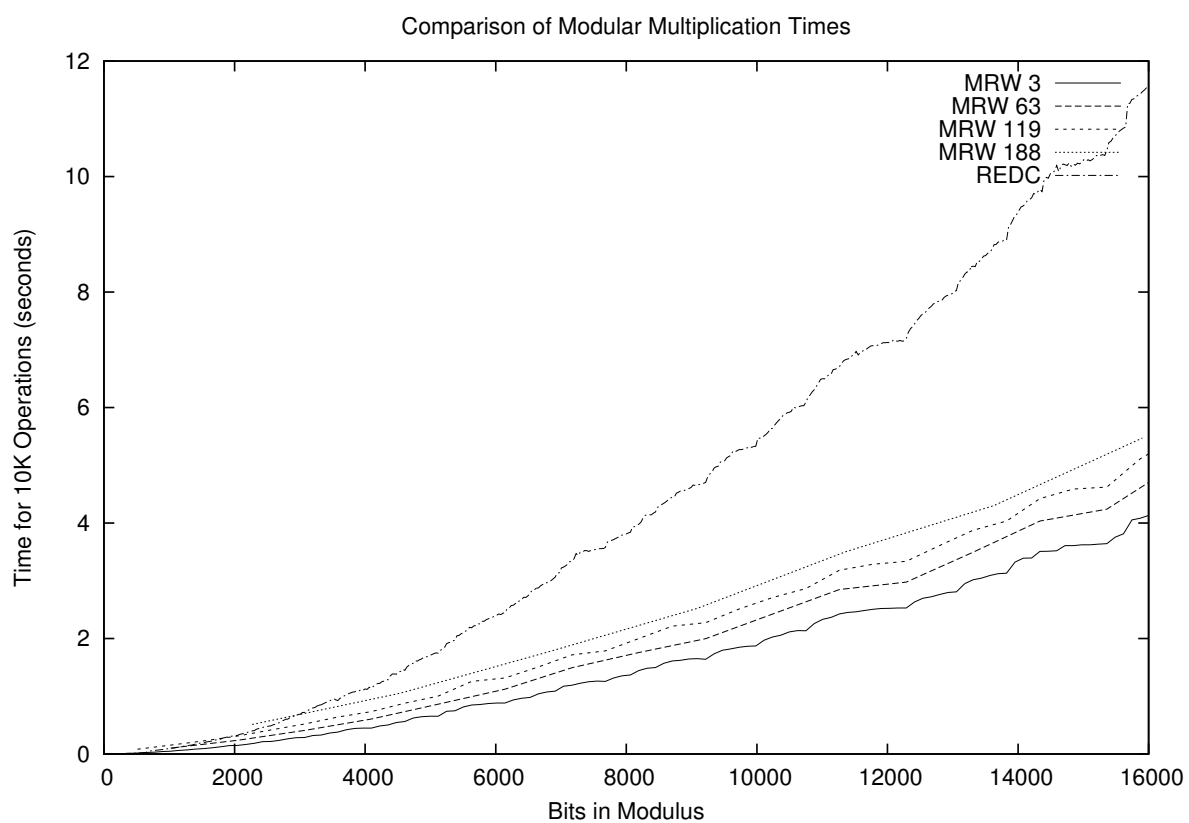


Figure 4.3: Modular multiplication times for selected MRW, and the REDC baseline.

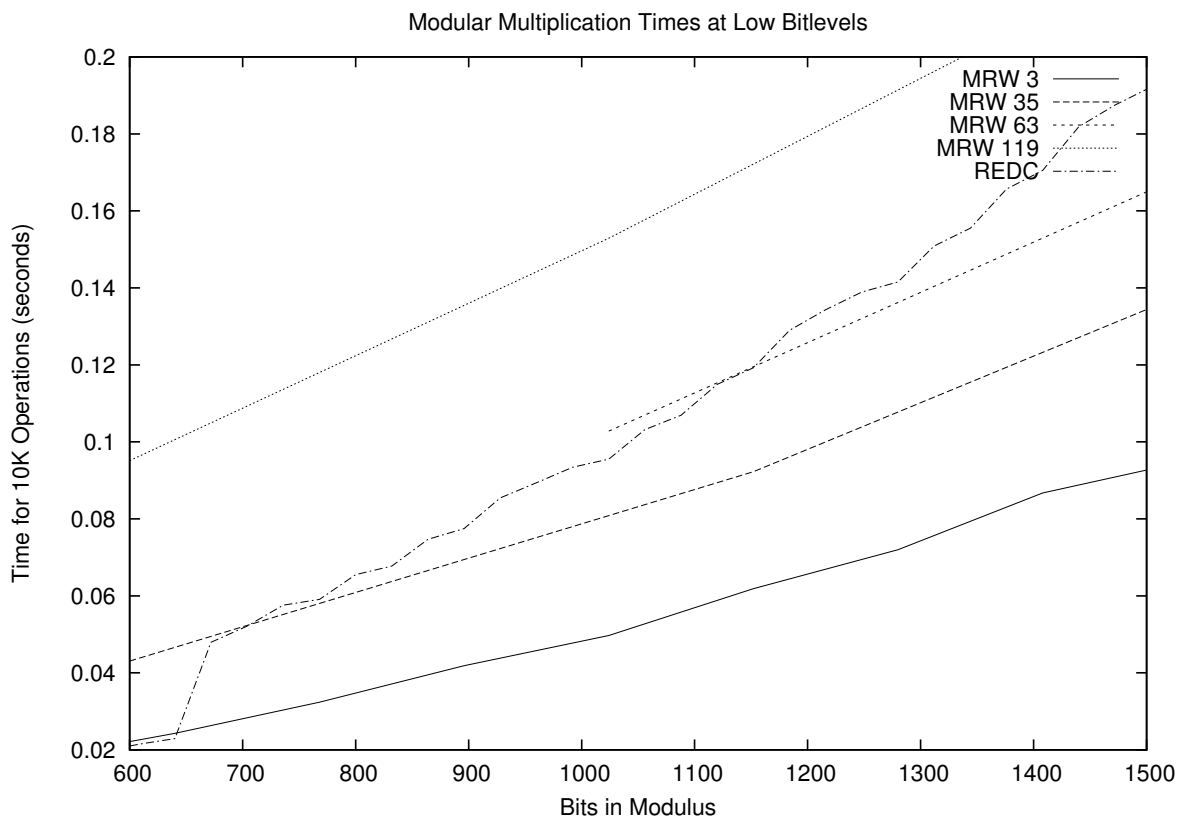


Figure 4.4: Modular multiplication times for selected MRW and the REDC baseline at bitlevels less than 1500.

Chapter 5

Pairing Computations

In this chapter we present Miller’s algorithm, which is used to compute both the Tate and the Weil pairings. We then discuss some related work used to improve efficiency of the algorithm. Next, we will review the role of generalized Mersenne numbers, used to improve the performance of pairing computations. We then focus on two specific cases (supersingular curves of embedding degree 1 and 2), discuss parameter selection and present the results of an implementation. The results from this section will be further discussed in a larger context in the following chapter.

5.1 Miller’s Algorithm

Miller’s algorithm originally appeared in [51], then more formally in [53] (some helpful examples can be found in [29]). This algorithm is the basis for computing both the Weil and Tate pairings described in Sections 2.4.1 and 2.4.2.

Recall the definition of the Weil pairing:

$$e_n(P, Q) = \frac{f_{n,P}(Q)}{f_{n,Q}(P)}$$

In order to compute the pairing, we must find the functions $f_{n,P}$ and $f_{n,Q}$ such that

$$\begin{aligned} \operatorname{div}(f_{n,P}) &= n(P) - n(\mathcal{O}) \text{ and} \\ \operatorname{div}(f_{n,Q}) &= n(Q) - n(\mathcal{O}) . \end{aligned}$$

First we must work with normalized functions (which will be defined momentarily). To do this, we start by fixing a uniformizer at P (§4.1 of [53] describes how). Once this is done, any rational function may be expressed as a Laurent series in u_p . A *Laurent series* is a power series in a variable where a finite number of terms are allowed to have negative exponents in said variable.¹ For example, take a function

¹The full definition can be found in the appendix of [53]. It differs from the definitions of a Laurent series for a complex function which can have an infinite number of negative terms, i.e. $f(z) = \sum_{n=-\infty}^{\infty} c_n(z-a)^n$ for $z \in \mathbb{C}$, where a is a constant and a collection of complex numbers c_n .

$f \in K(E)$ with $\deg(f) = n$. Then we can write

$$f(u_p) = au_p^n + bu_p^{n+1} + \dots .$$

We will denote the *leading term* by $lt(f) = au_p^n$ and the *leading coefficient* by $lc(f) = a$. When f does not have a zero or a pole at P , $lt_P(f) = f(P)$, and $lt_P(fg) = lt_P(f)lt_P(g)$. The leading coefficient will permit us to identify normalized functions; for $f \in K(E)$, f is *normalized* if $lc_{\mathcal{O}}(f) = 1$.

The advantage of normalized functions is that for every principal divisor \mathcal{D} there is a **unique** normalized function f such that $\text{div}(f) = \mathcal{D}$. Since the product of normalized functions is again normalized, from now on we will work exclusively with normalized functions.

Next, for two points $P, Q \in E$ call the normalized line through them $L_{P,Q}$. If $P = Q$, use the equation of the line tangent to the curve (cf. definition of addition law and Figure 2.1)). The divisor of this function² can be expressed neatly as

$$\text{div}(L_{P,Q}) = (P) + (Q) + (-(P + Q)) - 3(\mathcal{O}) .$$

Using this notation we could express the vertical line used during point addition as $L_{P+Q, -(P+Q)}$ (as in [53]). For greater simplicity, we write $V_{P,Q}$ for this vertical line. Lemma 1 of [53] gives the divisor of the rational function $L_{P,Q}/V_{P,Q}$;

$$\text{div}(L_{P,Q}/V_{P,Q}) = (P) + (Q) - (P + Q) - (\mathcal{O}) .$$

Now the idea is to define functions, $f_{i,P}$, inductively as

$$\begin{aligned} f_{0,P} &= 1 \\ f_{1,P} &= 1 \\ f_{m+1,P} &= f_{m,P} \frac{L_{P,mP}}{V_{P,mP}} . \end{aligned} \tag{5.1}$$

These functions have divisor

$$\text{div}(f_{m,P}) = m(P) - (m-1)(\mathcal{O}) - (mP) . \tag{5.2}$$

²The function that gives the equation of the line, that is $L_{P,Q}(x, y) = 0$.

Using them we can build up $f_{n,P}$, which will have the desired divisor

$$\begin{aligned}\operatorname{div}(f_{n,P}) &= n(P) - (n-1)(\mathcal{O}) - (nP) \\ &= n(P) - n(\mathcal{O})\end{aligned}$$

since $nP = \mathcal{O}$. Before forging an algorithm, we need two more properties about the functions $f_{m,P}$.

$$f_{i+j,P} = f_{i,P} \cdot f_{j,P} \cdot \frac{L_{iP,jP}}{V_{iP,jP}} \quad (5.3)$$

and

$$f_{ij,P} = (f_{i,P})^j (f_{j,iP}) = (f_{j,P})^i (f_{i,jP}). \quad (5.4)$$

Since Miller's algorithm relies on (5.3) heavily, we verify that the divisors of the left and right hand side are the same. On the left,

$$\operatorname{div}(f_{i+j,P}) = (i+j)(P) + (i+j-1)(\mathcal{O}) - (i+j)(P)$$

while on the right we first work out the divisors of the lines,

$$\begin{aligned}\operatorname{div}(L_{iP,jP}) &= (iP) + (jP) + (-(iP+jP)) - 3(\mathcal{O}) \\ \operatorname{div}(V_{iP,jP}) &= \operatorname{div}(L_{(i+j)P, -(i+j)P}) \\ &= (i+j)(P) + (-(i+j)(P)) + (-(i+j))(P) + (-(i+j))(P) - 3(\mathcal{O}) \\ &= (i+j)(P) + (-(i+j))(P) - 2(\mathcal{O})\end{aligned}$$

then the quotient,

$$\begin{aligned}\operatorname{div}\left(\frac{L_{iP,jP}}{V_{iP,jP}}\right) &= (iP) + (jP) + (-(i+j))(P) - (\mathcal{O}) \\ &\quad - ((i+j)(P) - (-(i+j))(P) + 2(\mathcal{O})) \\ &= i(P) + j(P) - (i+j)(P) - (\mathcal{O})\end{aligned}$$

and use (5.2) to get the complete right hand side:

$$\begin{aligned}\operatorname{div}\left(f_{i,P} \cdot f_{j,P} \cdot \frac{L_{iP,jP}}{V_{iP,jP}}\right) &= i(P) + j(P) - (i-1)(\mathcal{O}) - (j-1)(\mathcal{O}) - (i+j)(P) - (\mathcal{O}) \\ &= (i+j)(P) + (i+j-1)(\mathcal{O}) - (i+j)(P)\end{aligned}$$

which equals the left, so the functions have the same divisor.

The first of these two properties (Equation (5.3)) is similar to exponentiation (but with the extra quotient L/V , cf. $2^{i+j} = 2^i \cdot 2^j$). This similarity allows $f_{n,P}$ to be computed using any of the existing fast exponentiation methods, the most common

being the binary method³ where we iterate over the bits of n ([39] 4.6.3). This gives Miller’s algorithm for finding $f_{n,P}$ a running time of $O(\log n)$.

Although in Section 2.4.1, we defined the Weil pairing as $(-1)^n \frac{f_P(Q)}{f_Q(P)}$, to present the original version of Miller’s algorithm we must use the equivalent definition $(-1)^n \frac{f_P(D_Q)}{f_Q(D_P)}$. The divisor D_Q , must be equivalent to $(Q) - (\mathcal{O})$ and we will evaluate it at a function f_P which has divisor $(f_P) = (P) - (\mathcal{O})$. The denominator is defined analogously.

A simplified version of Miller’s algorithm was discovered later which computes the pairing as we had originally defined it. Proceeding chronologically, our first algorithm will find and evaluate $f_P(D_Q)$, and we will present the simplified version in §5.1.1.

In [29] (page 188, Lemma IX.6) it is shown that finding and evaluating $f_P(D_Q)$ is possible by computing $g = n(Q + S) - n(S)$ for an arbitrary S . This introduces “equivalence up to n -th powers”; $g(D)$ is only equivalent to $f_P(D_Q)$ up to n -th powers. After raising $g(D)$ to the power $(q^k - 1)/n$, we have $f_P(D_Q)$ exactly. Algorithm 2 gives pseudocode for Miller’s algorithm.

To complete the computation of the Tate pairing, the returned value $f \in \mu_n \in \mathbb{F}_{q^k}$ must be raised to the power $(q^k - 1)/n$. This produces a unique value. The Weil pairing requires two applications of Algorithm 2, and a division in \mathbb{F}_{q^k} .

The point S can simply be chosen at random (however this does not necessarily lead to the most efficient implementation). The only condition on S is that it must not be a point computed in the addition chain. If $S = T$ at some point during the algorithm, it will lie on one of L or V , causing $L(S) = 0$ or $V(S) = 0$. This happens with probability $O(\frac{\log p}{p})$ [16]. For large enough n , S may be fixed as $[i]P$, for i such that $[i]P$ does not appear in the addition chain of $[n]P$.

5.1.1 Related work

In this section we recap state of the art techniques which greatly reduce the computational cost of pairings, mainly through improvements to Miller’s algorithm. Much of the work discussed in this section is from Galbraith *et al.* [34], and Barreto *et al.* [13, 12, 14, 11]. Many optimizations are specific to fields of low characteristic \mathbb{F}_{2^m} and \mathbb{F}_{3^m} , and will not be applicable in our study. In this section we limit coverage to

³This method is sometimes also called “square and multiply” for multiplication or “double and add” for point multiplication on elliptic curves. See [28], IV.2.1 for the latter.

Algorithm 2 Miller's algorithm. Compute $f_P(D_Q)$.

INPUT : points $P \in E(\mathbb{F}_q)[n]$, $Q \in E(\mathbb{F}_{q^k})$, where P has order dividing n

OUTPUT: A value $f \in \mu_n \in \mathbb{F}_{q^k}$

- 1: Choose a suitable point $S \in E(\mathbb{F}_{q^k})$.
 - 2: $Q' \leftarrow Q + S$.
 - 3: $T \leftarrow P$.
 - 4: $m \leftarrow \lfloor \log_2(n) \rfloor - 1$, $f \leftarrow 1$
 - 5: **while** $m \geq 0$ **do**
 - 6: Calculate lines l and v for doubling T .
 - 7: $T \leftarrow [2]T$.
 - 8: $f \leftarrow f^{2 \frac{l(Q')v(S)}{v(Q')l(S)}}$.
 - 9: **if** the m -th bit of n is one **then**
 - 10: Calculate the lines l and v for addition of T and P .
 - 11: $T \leftarrow T + P$
 - 12: $f \leftarrow f^{\frac{l(Q')v(S)}{v(Q')l(S)}}$
 - 13: **end if**
 - 14: $m \leftarrow m - 1$
 - 15: **end while**
 - 16: **return** f
-

speedups applicable to all curves.

Improvements to Miller's Algorithm

First off, [34] notes that for the majority of cryptographic applications, $P \in E(\mathbb{F}_q)$ and $Q \in E(\mathbb{F}_{q^k})$. Consequently, most of the work in Miller's algorithm, stays in \mathbb{F}_q . Arithmetic in \mathbb{F}_{p^m} requires roughly $O(m^2)$ \mathbb{F}_p -multiplications; therefore smaller m can give large speedups. The randomly chosen point S can be chosen from $E(\mathbb{F}_q)$, in order to keep more of the arithmetic in the smaller field. These optimizations apply to computing $\langle P, Q \rangle$ but not $\langle Q, P \rangle$. Therefore, computing the Weil pairing, which is effectively $\frac{\langle P, Q \rangle}{\langle Q, P \rangle}$ costs significantly more than two computations of $\langle P, Q \rangle$ (see [29], p. 192, for details of this equivalence between pairings). While significant at lower levels, this difference is eventually dominated by the cost of exponentiation in the Tate pairing, and becomes less important at higher security levels.

A small and simple improvement removes divisions at each iteration of the algorithm. The divisions in steps 8 and 12 of Algorithm 2 can be collected up into a single division at the end.

As we've seen, the running time of Miller's algorithm is $O(\log n)$, so the choice of n , the order of the subgroup, can reduce the run time. However, n must be large enough to maintain the difficulty of the ECDLP. To reduce the number of additions (steps 9 to 13 in Algorithm 2) choose n with low Hamming weight. Some GM primes also have low Hamming weight, for example $2^{283} + 2^{142} + 1$ has Hamming weight 3. If the group order, $\#E$, has low Hamming weight, the authors of [34] note that we could choose $n = \#E$. In this case $(q^k - 1)/\#E$ will also have low Hamming weight, speeding the exponentiation. They also discuss the advantages of using a smaller subgroup, which would outweigh a larger group with low Hamming weight. The savings from using a smaller subgroup become even larger at higher security levels.

Another interesting part of [34] is the technique for representing extension fields. They create a tower of quadratic extensions of \mathbb{F}_{2^m} to represent $\mathbb{F}_{2^{4m}}$. Let $F = \mathbb{F}_{2^m}$.

$$F_1 = \frac{F[x]}{(x^2 + x + 1)} \approx \mathbb{F}_{2^{2m}} \quad \text{quadratic extension of } F$$

$$F_2 = \frac{F_1[y]}{(y^2 + (x + 1)y + 1)} \approx \mathbb{F}_{2^{4m}} \quad \text{quadratic extension of } F_1$$

General elements of F_2 can be written as $a + bx + cy + dxy$ with $a, b, c, d \in \mathbb{F}_{2^m}$. These elements can be multiplied with only 9 multiplications in F . A similar representation is employed for characteristic 3.

These fields are generalized in [41], and named “pairing friendly fields”. A field \mathbb{F}_{p^k} is *pairing friendly* if $p \equiv 1 \pmod{12}$ and k is of the form $2^i 3^j$. Using this representation gives multiplications in time $3^i 5^j m$ where m is the cost of a multiplication in \mathbb{F}_p .

The paper by Barreto *et al.* [14] begins by describing a procedure for constructing ordinary curves with arbitrary k . Until recently, the only curves known with k small enough to be useful for pairings were supersingular curves. Supersingular curves can have at most $k = 6$ (see [49]), but with non-supersingular curves, arbitrary k is possible. A random ordinary curve will have enormous k (shown in [10]), so one cannot simply choose curves at random. The construction can also be tweaked to find curves where n has low Hamming weight. An alternate algorithm for constructing ordinary curves with arbitrary k is also given in [29], Section IX.15.1. The references of this text present techniques to allow construction of curves with pairing friendly $k = 2^i 3^j$.

With respect to the pairing computation, Barreto *et al.* suggest improvements to Miller’s algorithm. Theorem 1 states that for linearly independent points $P \in E(\mathbb{F}_q)[n]$ and $Q \in E(\mathbb{F}_{q^k})$, the Tate pairing can be reduced to $\langle P, Q \rangle_n = f_P(Q)^{(q^k-1)/n}$ instead of $\langle P, Q \rangle_n = f_P(D_Q)^{(q^k-1)/n}$, with D_Q as defined above in the description of Miller’s algorithm. This allows the function f_P to be evaluated on a point rather than a divisor.

This result is then combined with another. To find the function f_i such that $(f_i) = i(P) - ([i]P) - (i-1)(\mathcal{O})$, Miller used the formula $f_{i+j} = f_i f_j \frac{l}{v}$. The authors provide an alternate formula for f_i ,

$$f_{i+j} = f_i(Q) f_j(Q) \frac{l(Q)}{v(Q)},$$

where l is the line between $[i]P$ and $[j]P$, v is the line between $[i+j]P$ and $-[i+j]P$ (vertical). This new formula leads to a simplified version of Miller’s algorithm, which gets used in practice.

A method is also given ([14], Theorem 3) by which the denominator $v(Q)$ in the computation of f can be eliminated without changing the value of the pairing.

Algorithm 3 Simplified Miller

 INPUT: Points $P \in E(\mathbb{F}_q)[n]$, $Q \in E(\mathbb{F}_{q^k})$

 OUTPUT: $f_P(Q)$

```

1:  $f \leftarrow 1$ 
2:  $T \leftarrow P$ 
3:  $m \leftarrow \lfloor \log_2(n) \rfloor - 1$ ,  $f \leftarrow 1$ 
4: while  $m \geq 0$  do
5:   Find the lines  $l$  and  $v$  for doubling  $T$ .
6:    $f \leftarrow f^2 \frac{l(Q)}{v(Q)}$ 
7:    $T \leftarrow [2]T$ 
8:   if the  $m$ -th bit of  $n$  is 1 then
9:     Find the lines  $l$  and  $v$  for adding  $T + P$ 
10:     $f \leftarrow f \frac{l(Q)}{v(Q)}$ 
11:     $T \leftarrow T + P$ 
12:   end if
13: end while
14: return  $f$ 

```

This simplification is limited to fields with characteristic greater than 3. Using this method, paring computation for $k = 2$ can be computed almost completely with \mathbb{F}_q arithmetic.

Implementation Related Work

We end this section by pointing out the scarcity of implementation timings available in the literature. There are limited timings from implementations in characteristic 2 and 3 in [11] and [34] at low security levels. To our knowledge there has not been a comprehensive comparison of the various possible implementations. This is likely due to the complexity of each implementation, coupled with the large number of small optimizations required for a proper comparison.

The analysis of Koblitz and Menezes [41] is the first to compare a large number of implementation options over a range of security levels. Although their analysis is only an estimate which makes some minor simplifying assumptions, it includes the majority of relevant optimizations. It provides the clearest overall picture to date, and our results should help clarify our understanding further.

5.2 Pairings with Generalized Mersenne Numbers

As we've seen, there are two uses for generalized Mersenne numbers in pairing computations. The first is fast modular reduction. If our elliptic curve E is taken over a field \mathbb{F}_p where p is a generalized Mersenne prime, the field arithmetic required to implement the operations in the curve group will be improved. Miller's algorithm consists of curve group operations and field arithmetic in the extension \mathbb{F}_{p^k} . Thus for pairing computations it is desirable that p be a generalized Mersenne number of low MRW. With the results of Chapter 4, we would like the MRW of p to be less than 200, and the exponents in the representation of p to be congruent to zero modulo the word size.

The second parameter we would like to be a generalized Mersenne number is n , the prime order of the subgroup used in Miller's algorithm. Let us consider the example where $n = 2^a - 2^b + 1$. Recall from §5.1 that Miller's algorithm incrementally computes functions $f_{n,P}$ for a point $P \in E$. First we compute $f_{2^a,P}$, $f_{2^b,P}$ which requires only doubling operations. Note that f_{2^a} can be computed from f_{2^b} using $a - b$ double

operations, hence both are computed with a doubling operations. Then we use the negation formula from [53],

$$f_{-2^b, P} = \frac{1}{f_{2^b, P} \left(\frac{L_{nP, -nP}}{V_{nP, -nP}} \right)}$$

and then the addition formula

$$f_{2^{a+(-2^b)}} = f_{2^a, P} f_{-2^b, P} \left(\frac{L_{2^a P, -2^b P}}{V_{2^a P, -2^b P}} \right).$$

The final addition can be done using formula (5.1). Hence $f_{n, P}$ is computed using only a doublings, instead of $\log_2(n) \approx a$ doublings and $\text{Hamming_weight}(n)$ additions. This generalizes to all GM primes, and their structure can be exploited to significantly increase the performance of Miller's algorithm.

At this point, some readers may have noticed that we are not concerned about the MRW of the prime n , nor does it matter that it aligns nicely with the wordsize. We present a related definition (from [63]). Any integer N can be represented as $N = \sum_{i=1}^w \epsilon_i 2^{e_i}$ for $\epsilon \in \{1, -1\}$, $e_i \in \mathbb{Z}$. The *weight* of an integer N is w , the number of terms in this sum. We note that for the computation of Miller's algorithm, it is possible to relax the condition on ϵ_i , allowing $\epsilon_i \in \mathbb{Z}$, but do not explore this possibility here. We call N a *low-weight* integer (or prime) if w is low⁴. Exactly how low depends on the application.

From the definitions given, a low-weight prime *is* a generalized Mersenne prime, and a GM prime *is* a low-weight prime (depending on one's interpretation of "low"). The key difference is that a low-weight prime may not be a practically useful GM prime, if it has high MRW. To differentiate, we will refer to a low-weight/GM prime as a *Solinas prime* if it has MRW low enough to be practical (MRW < 200). If we have the further condition that $e_i \equiv 0 \pmod{w}$ we call N a *word aligned* Solinas prime.

Before proceeding, we examine Solinas' modular reduction algorithm when the modulus is not word aligned, to gauge the importance of this property. Recall (from

⁴Note that this differs from the *Hamming weight* of an integer, which counts the number of 1s in the binary representation of the integer; a low weight integer can have a high Hamming weight (e.g. $2^n - 1$).

§4.3) that it works by representing an integer $m = f(2^q) < p^2$ as

$$m = \sum_{j=0}^{2d-1} A_j \cdot 2^{jq} ,$$

then re-casting the mod operation as additions/subtractions of the A_j .

The method of Solinas works best when q is a multiple of the wordsize w . When $w \nmid q$, just how “inconvenient” is this method, and is it still practical? We consider a few examples to explore the issue.

Example 1 The first example is when $q < w$ (example 4 from [41]). The polynomial f is

$$f(t) = t^{7737} - t^{7477} + t^{7216} + 1$$

which gives

$$p = f(2^2) = 2^{15474} - 2^{14954} + 2^{14432} + 1 .$$

Then we have $q = 2$ and $d = 7737$. To represent some $m < p^2$, we use the sum

$$m = \sum_{j=0}^{15473} A_j \cdot 2^{2j}$$

There are two problems with this. Each A_j is only 2 bits large, and there are 15473 of them. Also, the MRW of the polynomial is high; there will be 3524577 additions/subtractions required to reduce m (using the method presented in [74]). Recall that the MRW already counts operations involving contiguous A_j as a single operation, so we cannot combine any others.

Example 2 The second example considers the case when $q > w$. We take the polynomial

$$p(t) = t^3 - t^2 - 1$$

and choose $q = 50$. Then

$$p = p(2^{50}) = 2^{150} - 2^{100} - 1$$

To represent $m < p^2$, we write

$$m = \sum_{j=0}^5 A_j \cdot 2^{50j} .$$

Then there are up to 5 A_j . The MRW of $p(t)$ is 4, so although the A_j are misaligned, we need only use them for 4 operations.

Though we do not have supporting implementations and benchmarks to support it, we hypothesize that modular reduction by a misaligned modulus is still significantly faster than competing methods; provided the MRW is low. The misaligned A_j increase the cost of each addition/subtraction, but only slightly. The implementation complexity is also increased, and not supported by our code generation tool. For this reason we favor aligned Solinas primes.

With this motivation for using GM primes as parameters, we address the question: *How can we find elliptic curves $E(\mathbb{F}_p)$ where p is a GM prime, and $n|\#E$ such that n is a low-weight prime?* In addition, the sizes of n and p^k must be of appropriate size to provide the desired security.

5.3 Embedding Degree $k = 1$

As pointed out by Koblitz and Menezes [41], elliptic curves with embedding degree one have not been suggested for use in pairing based cryptography, the literature discusses them only briefly, and dismisses them without thorough consideration.

The only curve construction technique available was presented in [41], and uses supersingular curves. We review it now.

Chose a prime $p = A^2 + 1$. If $A \equiv 0 \pmod{4}$, then

$$E : y^2 = x^3 - x \tag{5.5}$$

if $A \equiv 2 \pmod{4}$, then

$$E : y^2 = x^3 - 4x . \tag{5.6}$$

Their Theorem 3 then proves $E(\mathbb{F}_p) \approx \mathbb{Z}_A \times \mathbb{Z}_A$ and hence $\#E(\mathbb{F}_p) = A^2 = p - 1$. Using these curves, we can select parameters by choosing $A = nh$, where n is prime, and $p = (nh)^2 + 1$ is also prime. There are a few further constraints:

1. n and p must have bitlengths large enough to provide the desired security;
2. n must be a low-weight prime;
3. p must be a Solinas prime as well, preferably word aligned.

The authors follow with three examples at the security levels 128, 192, 256. They appear in Table 5.1. Each of the Solinas primes given in the examples is misaligned, and the greatest common divisor of the exponents is 1 or 2. The examples also have a MRW too large to be practical. However, the choice of n in the examples is suitable for improving Miller's algorithm.

Example	$p(t)$	2^q	MRW(p)
2	$t^{3202} - t^{3121} + t^{3038} + t^{2947} - t^{2865} + t^{2690} + 1$	2	70049975387
3	$t^{8376} - t^{8333} + t^{8288} - t^{7991} + t^{7947} + t^{7604} + 1$	2	$> 10^{25}$
4	$t^{7737} - t^{7477} + t^{7216} + 1$	2^2	3524578

Table 5.1: Examples from [41] for embedding degree 1. The exact MRW of example 3 would require a careful implementation to compute, but by partially computing X we were able to determine that it is greater than 10^{25} .

The poor alignment and high MRW motivates a search. Our goal is to find parameters which are both aligned to our word size (32-bit) and have MRW small enough to be practical.

5.3.1 Parameter Search

For p to be an aligned Solinas prime, we must have

- $p = f(2^q)$ where $f \in \mathbb{Z}[t]$,
- f is monic and irreducible,
- f has odd, nonzero constant term,
- and $q \equiv 0 \pmod{32}$.

We will write the polynomial defining p as:

$$p(t) = t^d - c_1 t^{d-1} - \dots - c_{d-1} t - c_d \in \mathbb{Z}[t]. \quad (5.7)$$

As mentioned in the previous section; the improvement to Miller's algorithm requires only that n be a low-weight prime, we can ignore the alignment property. However, since $p = (nh)^2 + 1$, the form p will take depends on n as well, and so n must be chosen carefully. One possible approach is to choose n and h by using polynomials in $\mathbb{Z}[t]$, then p will have the desired form.

Proposition 1 *Let $p = (nh)^2 + 1$. If $n = n(2^q), h = h(2^q)$ for $h(t), n(t) \in \mathbb{Z}[t]$, and $q \in \mathbb{Z}$ then $p = p(2^q)$ for $p(t) \in \mathbb{Z}[t]$.*

Proof. Set $p(t) = [n(t)h(t)]^2 + 1 \in \mathbb{Z}[t]$. □

By the definition of a GM number $p(t)$ is monic, thus $n(t)$ and $h(t)$ must be monic as well. This provides one way to search for parameters. Ignoring MRW, our search can proceed by choosing $n(t)$ such that $n(2^q)$ is prime, choosing $h(t)$ and checking if the resulting $p(2^q)$ is prime.

What about the converse of Proposition 1? Can we give the search more flexibility when choosing n ? Consider $n = n(2^r)$ for $r \in \mathbb{Z}, r \not\equiv 0 \pmod{w}$. Then

$$p = [n(2^r)h(2^q)]^2 + 1$$

The polynomials $n(t), h(t)$ are monic, of degree d_1, d_2 for some $d_1, d_2 \in \mathbb{Z}$. The leading term of the product $n(2^r)h(2^q)$ is then $2^{d_1 r} \cdot 2^{d_2 q} = 2^{d_1 r + d_2 q}$, which, after the squaring becomes $2^{2d_1 r + 2d_2 q}$. The other exponents will also be sums and products of d_1, d_2, r, q . It is possible that they will have q as a common factor or all be congruent to zero mod w , but this is highly unlikely without special care in the choice of r, q and consideration of the degrees of the terms in $n(t), h(t)$. For fixed $n(t), h(t), r, q$ one could prove whether or not it is possible.

Let us consider an example, $n(t) = t - 1, r = 13, q = 32, h(t) = t$. Then

$$\begin{aligned} p(2^{32}) &= ((2^{13} - 1)2^{32})^2 + 1 \\ &= (2^{26} - 2^{14} + 1)(2^{64}) + 1 \\ &= 2^{90} - 2^{78} + 2^{64} + 1 \end{aligned}$$

In this example p is not prime but it illustrates how the exponents are determined additively.

Our search for primes $p = (nh)^2 + 1$ will proceed by finding $n = n(2^q), h = h(2^q)$ and $p = p(2^q)$ where $q \equiv 0 \pmod{32}$. It remains an open problem to make use of the fact that only p must be an aligned Solinas prime, n can be any low-weight prime.

The following steps describe our search:

1. Generate a random irreducible polynomial $n(t) \in \mathbb{Z}[t]$
2. Check if $n = n(2^{32j})$ is prime for all $j > 0 \in \mathbb{Z}$ such that $\text{bitlength}(n) < 1000$.

3. If n is prime, then compute $p(t) = (n(t)h(t))^2 + 1$ for varying values of $h(t)$.
4. if $p(t)$ is irreducible, evaluate $p(2^{32j})$ and check if it is prime.

For “varying $h(t)$ ” we heuristically chose $h(t) = t^i, t^i \pm t^m$ for all i, m that kept the bitlength of p less than 15000. The basis for this heuristic is an attempt to keep $p(t)$ “simple”. That is, we would like to keep the degree of $p(t)$ low, and keep the coefficients as small as possible, to produce primes with MRW low enough to be practical.

This approach was implemented with the PARI/GP scripting language. Some parameters were found, however the search ignored the MRW. All of the primes found had a MRW too high to be practical. We were also not searching for n, p of a specific relative bitlengths. A sampling is listed in Table 5.2.

$n(t)$	$h(t)$	2^q	MRW($n(t)$)	MRW($p(t)$)
$t^{13} + 3t^4 + 2t^3 + t^2 + 3t + 1$	t^2	2^{32}	63	44263
$t^3 - t - 1$	$t^8 - t^3$	2^{64}	3	74962
$t^6 + 3t^3 + 2t^2 + 1$	$t^2 + t$	2^{32}	29	91231
$t^9 + 2t^4 + 3t^3 + 2t + 3$	t^3	2^{32}	47	112080
$t^{16} - t - 1$	$t^{101} - t^{38}$	2^{32}	3	9804870
$t^{16} - t - 1$	$t^{135} - t^{30}$	2^{32}	3	245350358
$t^{19} - t^{16} - 1$	$t^{108} - t^{29}$	2^{32}	8	424385648767
$t^{24} - t^3 - 1$	$t^{114} - t^{47}$	2^{32}	3	1206254
$t^9 + 2t^5 + 4t^4 + 4t^3 + t^2 + t + 1$	t^2	2^{32}	117	2054435
$t^3 - t - 1$	$t^{45} - t^{22}$	2^{64}	3	253220781306736
$t^3 - t - 1$	$t^{45} - t^{28}$	2^{64}	3	256139034781873

Table 5.2: List of polynomials $n(t)$ and $h(t)$ such that $n(2^q)$, and $p(2^q)$ are both prime where $p(t) = [n(t)h(t)]^2 + 1$.

5.3.2 Impracticality of $k = 1$

Without having found suitable parameters for implementation we now argue that the curve construction technique we attempted used is impractical. In this section we attempt to explain why our search failed. With the methods presented in this chapter, it is impractical to use a curve with $k = 1$ over a field that is a GM prime.

The search of the previous section did find parameters, but none with an acceptably low MRW. The examples of Table 5.2 suggest that $\text{MRW}(p(t))$ is always much

larger than $\text{MRW}(n(t))$. If this is the case in general, then there is a low probability of a randomly chosen $n(t)$ and $h(t)$ leading to $p(t)$ with low MRW.

We wish to explore the relationship between $\text{MRW}(p)$ and $\text{MRW}(n)$. Figures 5.1 and 5.2 plot the MRW of $n(t)$ on the x -axis, and the corresponding weight of $p(t) = [n(t)t]^2 + 1$ on the y -axis. The choice $h(t) = t$ was made since it would result in $p(t)$ of lowest degree and coefficients. The monic polynomials $n(t)$ were chosen at random

These two figures show that the MRW of $p(t)$ is roughly quadratic as a function of the MRW of $n(t)$. Figure 5.2 limits the y -axis to 2000, to show just how sharply the MRW of p increases. This leaves few choices for $n(t)$ such that $p(t)$ has low MRW. Then, amongst these few, we search for a case where n and p are simultaneously prime, and of appropriate size for our application.

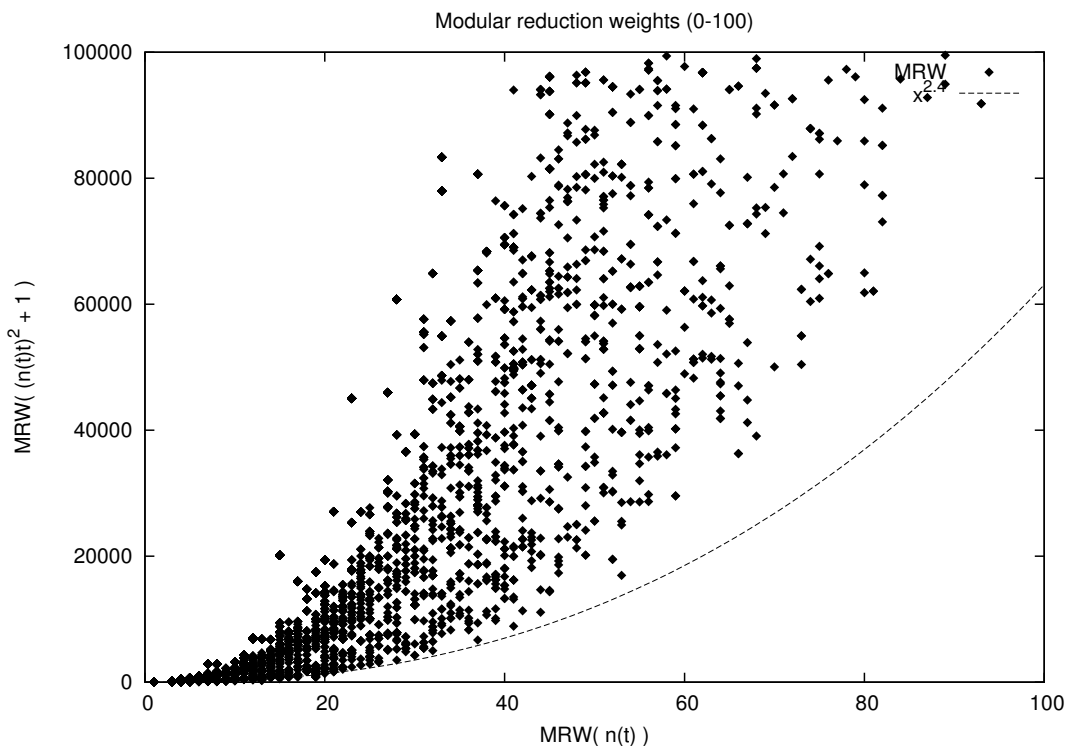


Figure 5.1: Plot of $\text{MRW}([n(t)t]^2 + 1)$ for $n(t)$ with modular reduction weight from 0 to 100.

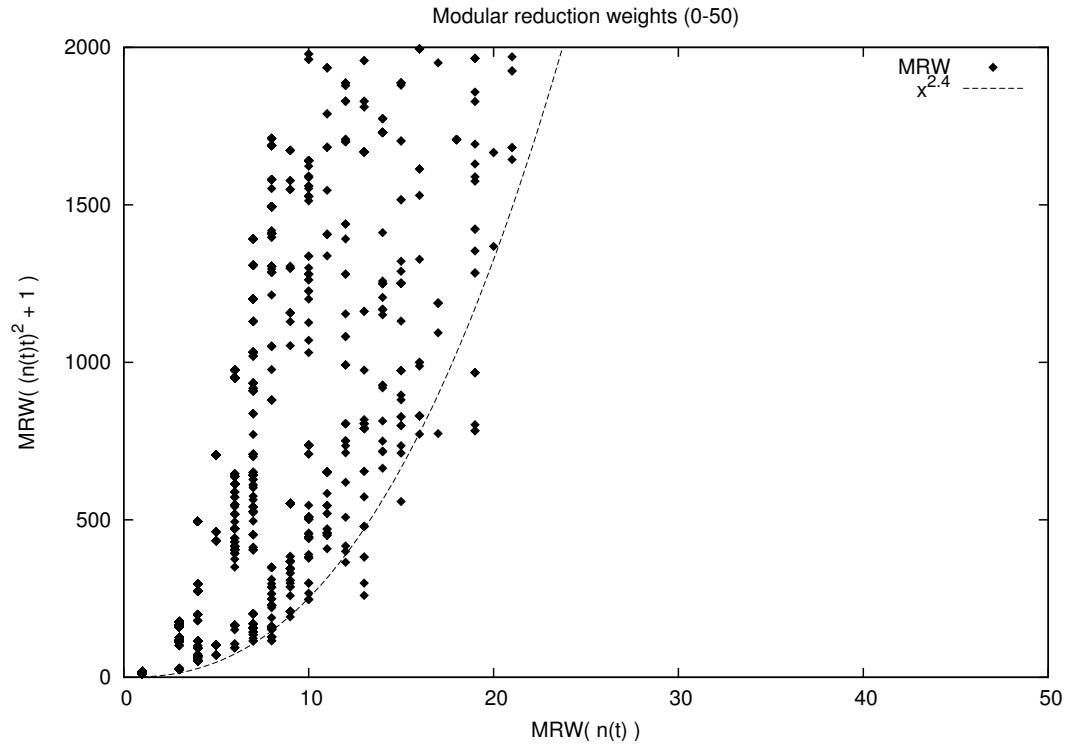


Figure 5.2: Plot of $MRW([p(t)t]^2 + 1)$ for $n(t)$ with modular reduction weight from 0 to 50.

5.3.3 Conclusions for $k = 1$

Without an efficient way of finding primes n and p with low MRW, the modular reduction operation in the field cannot be improved by using GM primes. It may be possible to find parameters when we do not require the word aligned property, but we have not investigated this possibility. However, n can still be chosen, and its structure used to render Miller's algorithm more efficient.

5.4 Embedding Degree $k = 2$

Curves with embedding degree 2 are more popular for pairing-based cryptography, and were even used as an example in the first IBE paper [16]. Again, we will present the curve construction method of [41]. We use the same primes, n and p , however this time $p = nh - 1$, and $4|h$. Then we can use the supersingular curve

$$E : y^2 = x^3 - 3x$$

when h is not divisible by 3. If h is divisible by 3, then let E be

$$y^2 = x^3 - 1$$

instead. The order in either case is $\#E(\mathbb{F}_p) = p + 1 = nh$.

In table 5.3 we list the examples given in [41] for embedding degree two. Although the prime p is not word aligned, examples 6 and 9 have low enough MRW to still be practical.

Example	$p(t)$	2^a	MRW(p)
6	$t^{520} + t^{363} - t^{360} - 1$	2	31
7	$t^{1582} + t^{1551} - t^{1326} - 1$	2	82637
8	$t^{4231} - t^{3907} + t^{3847} - 1$	2	10571
9	$t^{3882} - t^{3352} - 1$	2^2	9

Table 5.3: Examples used in [41] for embedding degree two. At the 80 and 256-bit security levels (Examples 6 and 9) the MRW is low enough to allow practical implementation.

5.4.1 Parameter Search

We conducted a search for aligned primes to simplify implementation. This search attempted to find primes where $n(t) = t^a \pm t^b \pm 1$, using the strategy described in the previous section for embedding degree one. Since the form of p is simpler, $nh - 1$ as opposed to $(nh)^2 + 1$, we expected better results.

This was the case, and we were able to find parameters. The numbers in Table 5.4 are the result of searching for $n(t)$ of the form $t^a \pm t^b \pm 1$, then $h(t) = t^j$ or $h(t) = t^j - t$.

Parameters Used

In Table 5.4 the first 12 numbers have $\text{MRW} < 200$. Examples with low MRW, are abundant at the 80 and 128-bit levels, but were scarce at 192 bits. None were found at the 256-bit level. In Table 5.5, we list the parameters that will be used in our implementation.

We are not claiming that aligned parameters at the 256-bit security level can not be found, simply that it is difficult with the methods presented. When one drops the

$n(t)$	$h(t)$	q	MRW(n)	MRW(p)	b_n	b_p
$t^{14} - t^{17} - 1$	t^3	32	3	18	448	545
$t^7 - t^3 + 1$	$t^7 - t$	32	4	20	224	448
$t^{21} - t^6 + 1$	$t^{24} - t$	32	4	23	672	1440
$t^7 - t^2 + 1$	$t^2 - t$	96	4	28	672	864
$t^{13} - t^7 + 1$	$t^4 - t$	64	6	36	832	1088
$t^{24} - t^3 - 1$	$t^{27} - t$	32	3	39	768	1632
$t^{14} - t^7 - 1$	t^{25}	32	3	51	448	1248
$t^3 - t + 1$	$t^{10} - t$	32	4	77	96	416
$t^{24} - t^3 - 1$	$t^{72} - t$	32	3	123	768	3072
$t^{10} - t^9 + 1$	$t^3 - t$	32	19	125	320	416
$t^{19} - t^{16} - 1$	$t^{24} - t$	32	8	148	608	1376
$t^{10} - t^9 + 1$	$t^{16} - t$	32	19	193	320	832
$t^7 - t^2 + 1$	$t^{29} - t$	96	4	201	672	3456
$t^{14} - t^7 - 1$	$t^{50} - t$	32	3	476	448	2048
$t^{31} - t^{26} + 1$	$t^{109} - t$	32	14	2588	992	4480
$t^3 - t + 1$	$t^{24} - t$	32	4	4401	96	864
$t^6 - t^2 - 1$	$t^{47} - t$	32	3	8828	192	1696
$t^{19} - t^{16} - 1$	$t^{200} - t$	32	8	207224229	608	7008

Table 5.4: Some polynomials producing primes $n = n(2^q)$ and $p = n(2^q)h(2^q) - 1$, sorted by MRW(p). b_n and b_p are the number of bits in n, p .

restriction that the primes be word-aligned, there is the example from Table 5.3 of MRW 9, and others are presumably abundant.

Level	$n(t)$	$h(t)$	q	MRW(p)	b_n	b_p
80-bit	$t^3 - t - 1$	$t^5 + t^3 - t^2$	64	36	192	512
80-bit	$t^3 - t - 1$	$t^8 + t^6 + t^5$	64	44	192	704
128-bit	$t^{16} - t + 1$	$t^{34} - t$	32	33	512	1600
192-bit	$t^{15} - t^2 - 1$	$t^{56} - t^{17}$	64	188	960	4544

Table 5.5: Parameters used in our implementation.

5.4.2 Implementation and Experiments

In this section we describe an implementation, and benchmarks we performed in order to determine the impact of using our aligned GM prime parameters. We were fortunate that an implementation of the Boneh and Franklin IBE scheme was available. It has now been absorbed into the PBC Library [3]. Written by Ben Lynn, the Stanford IBE software computes both the Tate and Weil pairings, and has implementations

Miller's algorithm, one of which can handle GM primes.

The IBE software is implemented to use the curve $y^2 = x^3 + 1$, with $p \equiv 2 \pmod{3}$. The order of this curve[†] is $\#E(\mathbb{F}_{p^k}) = p + 1$, and the embedding degree is the smallest k such that $n|p^k - 1$. Since E is supersingular $k \leq 6$, and we can easily determine k . If $k = 1$, $n \nmid nh - 2$, but when $k = 2$, after substituting $p = nh - 1$ and expanding we have $n|n(nh^2 - 2h)$, hence the embedding degree is $k = 2$. The curves are similar enough that we can use the same parameters as we would with the curves given in [41], by choosing $p = nh - 1$.

The PBC library implements arithmetic in \mathbb{F}_{p^2} using the GMP library. We will replace the calls to `gmp_mod()` with our routines to perform Solinas reduction. We will not include the REDC reduction algorithm in this set of comparisons. The results of the previous chapter show that the REDC and GMP modular reduction functions have very similar performance at the bitlevels under consideration.

Miller's algorithm is used to implement the Tate and Weil pairings. Along with a plain version of Miller's algorithm, a faster variant is implemented to take advantage of low-weight, $n = 2^a \pm 2^b \pm 1$.

The primes p, n that will be used in our experiments appear in Table 5.5. Using these parameters, we will compare the performance of the variants of Miller's algorithm. Then we compare the time required to compute two versions of the Tate/Weil pairing. The first uses a straightforward implementation of Miller's algorithm while the second takes advantage of the special structure of n .

The runtime of the plain version of Miller's algorithm is greatly affected by the Hamming weight of n . The Hamming weights of our parameters are

- 80-bit level: n is 192-bits and has a Hamming weight of 129,
- 128-bit level: n is 512-bits, and has Hamming weight 481,
- 192-bit level: n is 960-bits, and has a Hamming weight of 959.

In all of our examples, n has high Hamming weight. As we saw at the beginning of this chapter, the performance of the low-weight version of Miller's algorithm is comparable to Miller's algorithm on a prime of low Hamming weight. Because of

[†]This is a class I supersingular curve, see [49].

this, our comparison of the Miller algorithms also gives insight to the importance of the Hamming weight in the general case, i.e. when n is not low-weight.

5.4.3 Results

We now present the timing results of our implementation of Miller’s algorithm, and the Tate and Weil pairings with embedding degree two.

In the timings presented below, each time is the average of 5 runs.⁵ (Each run computes 1 pairing, or runs Miller’s algorithm once). The input points used to time the algorithms were chosen randomly at the outset. The labels (of Tables 5.6 – 5.12) used should be interpreted as follows: “Fast” Miller/Reduction refers to the algorithms which use the structure of the primes, while the “Plain” versions work on general inputs. The top left corner is the time required when both “Fast” methods are applied, while the bottom right is when both “Plain” methods are used.

Miller’s Algorithm

We present the timings in Tables 5.6, 5.7, 5.8 and 5.9. First we comment on the effect of using a low-weight prime n in Miller’s algorithm. The timings observed for Miller’s algorithm reveal that this optimization provides an average speedup of 61%. This figure is higher than it would be in the average case, due to the high Hamming weights of the low-weight primes in our parameters. If we assumed that the average prime would have a Hamming weight leading to performance in the middle of the “Fast” and “Plain” Miller algorithms, the speedup would be a more modest 44%. Though not exact, this approximately quantifies the benefits of choosing n to be of low Hamming weight, as well as the a GM prime.

With respect to the Solinas reduction algorithm, the difference is smaller. In the results of the Chapter 4, we saw that the speedup of Solinas arithmetic increases with the bitlevel. This explains the speedup increase from 11% at 80-bit to 21% at 128-bit to 22% at 192-bit.

The combination of both special algorithms produced an observed speedup of 67%, 69% and 68% at the 80, 128 and 192-bit security levels.

⁵Again, the observed standard deviation of the runs was insignificant.

80-bit	Fast Reduction	Plain Reduction
Fast Miller	0.0218	0.0238
Plain Miller	0.0564	0.0664

Table 5.6: Miller’s algorithm at the 80-bit security level (MRW 36, $b_p = 512$).

80-bit	Fast Reduction	Plain Reduction
Fast Miller	0.0374	0.1026
Plain Miller	0.0385	0.1063

Table 5.7: Miller’s algorithm at the 80-bit security level (MRW 44, $b_p = 704$).

Tate vs. Weil

It was suggested in [41] that at higher security levels, it may be more efficient to use the Weil pairing instead of Tate. Recall that the Weil pairing requires two computations of Miller’s algorithm and one division in \mathbb{F}_{p^k} . By contrast, the Tate pairing requires only one use of Miller’s algorithm, which is then raised to the power $(p^k - 1)/n$. The cost of this final exponentiation in the Tate pairing becomes significant at higher security levels, the analysis of [41] suggests that the Weil pairing is the faster choice starting at the 192-bit level.

Indeed, the timings observed in this work support this conclusion. In Tables 5.10, 5.11 and 5.12, the time required by the Weil and Tate pairings appear for comparison. The faster time is in bold face, except at the 192-bit security level where they are nearly identical.

Effect of Curve Selection

When choosing an elliptic curve $E : y^2 = x^3 + ax + b$ the cost of computing $[2]P$ for $P \in E$, may differ by an addition (following the group operation in §2.2), and the cost of computing $P + Q$ is independent of a, b ($Q \in E, P \neq Q$). The time required for curve addition is dominated by multiplications of the point’s coordinates. The number of curve group operations in Miller’s algorithm depends only on n , and the number of times the algorithm is run to compute a pairing is fixed across all curves. For these reasons any a, b giving a curve with the correct properties (GM prime parameters and embedding degree) can be considered representative.

128-bit	Fast Reduction	Plain Reduction
Fast Miller	0.3038	0.3828
Plain Miller	0.7878	0.9995

Table 5.8: Miller’s algorithm at the 128-bit security level.

192-bit	Fast Reduction	Plain Reduction
Fast Miller	3.5210	4.4777
Plain Miller	8.5172	11.1640

Table 5.9: Miller’s algorithm at the 192-bit security level.

5.5 Conclusions

Our results confirm that the improvement GM primes provide to Miller’s algorithm is quite significant. The specific advantage depends on the comparison. A low Hamming weight n should provide similar performance, while one with high Hamming weight can be up to 61% slower. As for Solinas arithmetic, at higher security levels it provides a solid 20% speedup.

We have also confirmed prior analysis suggesting that the Weil pairing outperforms the Tate pairing at security levels of 192-bits and greater. This result applies to supersingular curves of embedding degree two.

80-bit (MRW 36)	<u>Fast Reduction</u>		<u>Plain Reduction</u>	
	Tate	Weil	Tate	Weil
Fast Miller	0.0340	0.0466	0.0380	0.0510
Plain Miller	0.0725	0.1208	0.0849	0.1362

Table 5.10: Weil/Tate comparison at the 80-bit security level.

128-bit	<u>Fast Reduction</u>		<u>Plain Reduction</u>	
	Tate	Weil	Tate	Weil
Fast Miller	0.6251	0.6915	0.8018	0.8338
Plain Miller	1.1902	1.6790	1.4919	2.0706

Table 5.11: Weil/Tate comparison at the 128-bit security level.

192-bit	<u>Fast Reduction</u>		<u>Plain Reduction</u>	
	Tate	Weil	Tate	Weil
Fast Miller	8.3680	8.34386	11.3332	10.3823
Plain Miller	14.6979	18.2940	19.3349	23.8012

Table 5.12: Weil/Tate comparison at the 192-bit security level.

Chapter 6

Implications for Pairing-Based Cryptography

This chapter will apply the results of Chapters 4 and 5 to discuss some issues important to pairing-based cryptography. The first section will discuss and improve upon analysis from [41], which will put our results in a larger context by comparing a variety of security levels and embedding degrees. In the second section we assess the impact of an improvement to the number field sieve for finding discrete logarithms in \mathbb{F}_{p^k} when p has low weight. The chapter ends with some recommendations.

6.1 Time Comparisons

The work of Koblitz and Menezes presents an analysis of the cost of computing the Tate and Weil pairings with embedding degrees[†] 1, 2(ss), 2(ns), 4, 6, 12, 24 ([41], Table 3). For supersingular curves of embedding degree 1 and 2 curve construction techniques allow one to choose p and n such that both are GM primes. At this time however, there is no known method to produce *practical* parameters for embedding degree 1, as discussed in §5.3.2.

After a careful analysis of the number of operations required to compute each pairing, formulas are given to count the number of operations required for each bit of n . The table gives the cost of computing the faster of the two pairings at each embedding degree and security level. The costs are stated as $aT(b)$ where a is the number of multiplications and $T(b)$ is the time required to multiply in \mathbb{F}_p where p is a b -bit prime. At embedding degrees 1 and 2, when p is a Solinas prime, the time is denoted $\tilde{T}(b)$.

We should say a few words about the analysis leading to the cost estimates. It proceeds by comparing the operations at each of the b_n iterations of Miller's algorithm (one iteration for every bit of n). Then the assumption is made that n will be a Solinas prime or of low weight. This allows the analysis to ignore add operations, and to only

[†]“ss” = supersingular; “ns” = nonsupersingular. 4-24 are non-supersingular.

consider the cost of doubling at each bit of n . This is reasonable, since there will be a negligible amount of adds relative to doublings.

When $k \geq 2$, the estimates also include speedups that come from working in subfields. If $\langle P \rangle$ is chosen carefully, a significant amount of arithmetic can be done in $\mathbb{F}_{p^{k/2}}$ instead of \mathbb{F}_{p^k} . There is another optimization when $k \geq 2$, which reduces the work required in the final exponentiation of the Tate pairing.

With respect to the actual operations, they are all stated as multiplications in \mathbb{F}_p . This is achieved by ignoring additions, working out the number of \mathbb{F}_p multiplies required for two \mathbb{F}_{p^k} operands¹ and assuming squaring is equivalent to multiplication. Experiments with the GMP library support this last assumption; for the bitlevels under consideration squaring was only a constant 10% faster.

Once these costs have all been determined, comparisons can be made at the various embedding degrees and security levels, but one unknown remains. Without knowledge of how $T(b)$ compares to $\tilde{T}(b)$, it is not possible to compare a time $a_1T(b_1)$ to $a_2\tilde{T}(b_2)$.

Security (bits)	80	128	192	256
bitlength of p^k	1024	3072	8192	15360
$k = 1^*$	$27T(1024)$	$33T(3072)$	$42T(8192)$	$42T(15360)$
$k = 1$	$27\tilde{T}(1024)$	$33\tilde{T}(3072)$	$42\tilde{T}(8192)$	$42\tilde{T}(15360)$
$k = 2(\text{ss})$	$22\tilde{T}(512)$	$28\tilde{T}(1536)$	$36\tilde{T}(4096)$	$36\tilde{T}(7680)$
$k = 2(\text{ns})$	$22T(512)$	$28T(1536)$	$36T(4096)$	$36T(7680)$
$k = 6$	$58T(171)$	$77T(512)$	$108T(1365)$	$133T(2560)$
$k = 12$		$203T(256)$	$296T(683)$	$365T(1280)$
$k = 24$				$1049T(640)$

Table 6.1: Table 3 from [41], pairing computation cost for each bit of n . An additional row $k = 1^*$ was added for the case when Solinas reduction is *not* used.

We reproduce Table 3 of [41] as Table 6.1. Using the timings in Chapter 4, we can replace the $T(b)$ and $\tilde{T}(b)$ with the actual times observed. This will allow comparison of all parameters considered, as all costs will have the same units. It will also show the relative performance of $k = 1$. This case will be split into two, when Solinas reduction is used, and when standard modular reduction is used. We restate that this is a “what-if” analysis, to determine what performance is possible should a curve construction method that yields practical parameters be found.

¹There are optimizations possible here as well, using the Karatsuba and Toom-Cook methods if \mathbb{F}_{p^k} is “pairing-friendly” (see §5.1.1). This angle is too tangential to be described here, but is clearly

Security (bits)	80	128	192	256
bitlength of p^k	1024	3072	8192	15360
$k = 1^*$	2.5793	23.7270	167.0760	439.1142
$k = 1$	1.5930	10.2399	63.2352	158.9700
$k = 2(\text{ss})$	0.5412	3.0492	17.2152	47.6676
$k = 2(\text{ns})$	0.7662	5.4656	40.9104	122.2740
$k = 6$	0.6356	2.6819	17.8848	65.1567
$k = 12$		2.9029	15.4304	51.6475
$k = 24$				52.0304

Table 6.2: Updated Table 3 from [41]. $T(b)$ replaced with faster of REDC, GMP. $\tilde{T}(b)$ replaced with timing at MRW 15. For $b \not\equiv 0 \pmod{32}$, the nearest larger timing was used (ie. $171 \rightarrow 192$, $1365 \rightarrow 1376$, $683 \rightarrow 704$).

Table 6.2 shows good performance in the supersingular $k = 2$ case. The advantage is most significant at the highest security level, where the benefits of Solinas reduction are greatest.

The times for embedding degree 1 with Solinas reduction are on average about 3 times slower than $k = 2(\text{ss})$. It was clear that $k = 1$ would be slower, however it was not known how it would compare to the others (this appears as “Open Problems 7” in [41]). The relative performance based on the analysis of [41] and our timings appear in Table 6.2. It was the slowest of all, even in the (hypothetical) estimates using Solinas reduction. With standard reduction, it becomes extremely slow (5 to 9 times slower than $k = 2(\text{ss})$). With the large difference between the two $k = 1$ cases, should an application have a reason to favor $k = 1$, effort should be invested to find a curve construction technique which will allow the use of a Solinas prime characteristic.

Based on the original table (our table 6.1) the authors suggest that for nonsupersingular $k \geq 2$ to use larger k for best performance. This is generally supported by the times in Table 6.2. However, it does not appear that larger is *always* better, there are two counter examples: $k = 12$ is slower than $k = 6$ at 128-bits of security and $k = 12$ is faster than $k = 24$ at the 256-bit security level. Since these timings are partially estimates, they are inconclusive, and unfortunately more experimentation is necessary. In addition, the new methods of [36] are applicable for larger k and it may always be faster to use the Tate pairing.

described in [41].

There is an inverse relationship between the embedding degree and the advantage of reduction modulo a Solinas prime. This is caused by the decrease in operand size at larger k . To achieve the same level of security in \mathbb{F}_{r^k} as in \mathbb{F}_p (for primes p, r) r must only be b_p/k bits. Table 6.1 shows the number of bits in p as k increases. At the 256-bit security level, $k = 2$ has $b_p = 7680$, and we can realize a 61% speedup if p is Solinas with MRW 15 (30% for MRW 35). By contrast, for $k = 24$, $b_p = 640$, the speedup provided by Solinas arithmetic drops to 34% at MRW 15, and offers no speedup for MRW 35. So as k increases, the benefit of choosing p to be a Solinas prime decreases, providing less motivation for curve construction that sets p to be Solinas prime with large k . Such a search would also have to find primes of much lower MRW as the acceptable upper limit drops considerably from 200 (down to about 15 in the example above).

Finally, we remind readers that the estimates in Tables 6.1 and 6.2 are precisely that, *estimates*. When we compare the implementation timings of chapter 5 to the estimates presented here, the trends are identified correctly, which helps validate this analysis. The agreement between the analysis and our observations with respect to the crossover point in the cost of pairings also lends support to the accuracy of the analysis. However, differences between quantities are not precise, and this should be understood when making conclusions.

6.2 Vulnerability of Generalized Mersenne Primes to the NFS

Often in cryptography, efficiency enhancements have an impact on security. Take for example, the case of \mathbb{F}_{2^m} . Arithmetic is cheaper, however, discrete logs can be solved more easily (see §3.1). With low weight primes, it was feared that it may be possible to use the special number field sieve (SNFS) instead of the number field sieve (NFS) to solve discrete logarithms in \mathbb{F}_{p^k} . We do not review these algorithms here, but borrow a quote from Crandall and Pomerance [25] to describe them: “These methods may be thought of as grand generalizations of the index-calculus method, and what makes them work is a representation of group elements that allows the notion of smoothness”. Some relevant references for using the NFS/SNFS to solve discrete logs are [35, 64, 4, 5, 62].

The running time required by the SNFS to solve discrete logs in a $2b$ -bit field

is roughly the time required by the NFS in a b -bit field [41]. Since the security of pairing based schemes depends on the difficulty of the DLP in \mathbb{F}_{p^k} , this is relevant to our study (see §2.5.1, “Relevant Presumably Hard Problems”).

A recent preprint by O. Schirokauer [63] investigates improvements to the number field sieve that can be made when p has low weight. He estimates (and gives examples) that a b_{p^k} -bit field only provides an estimated $(b_{p^k} - \gamma)$ -bits of security. In other words, discrete logs in the field of size b_{p^k} are only as difficult as discrete logs in a field of size $b_{p^k} - \gamma$.

We recall the examples (re-used from examples from [41]) where the modified NFS provides an improvement.

#	p	k	b_{p^k}	$b_{p^k} - \gamma$	% chg.
1	$2^{8376} - 2^{8333} + 2^{8288} - 2^{7991} + 2^{7947} + 2^{7604} + 1$	1	8376	7470	11
2	$2^{15474} - 2^{14954} + 2^{14432} + 1$	1	15474	13180	14
3	$2^{520} + 2^{363} - 2^{360} + 1$	2	1040	880	16
4	$2^{1582} + 2^{1551} - 2^{1326} - 1$	2	3162	2250	29
5	$2^{4231} - 2^{3907} + 2^{3847} - 1$	2	8462	5850	31
6	$2^{7746} - 2^{6704} - 1$	2	15492	9770	37

Table 6.3: The examples used in [63]. b_p is the size of the field \mathbb{F}_{p^k} in bits. $b_p - \gamma$ is the estimated security in bits provided. The last column shows the percentage decrease in the security provided.

The decrease in security is not as large as the SNFS could have potentially been, but is still significant. Two things are apparent from Table 6.3 and the comments made by the author. The modified NFS works faster with larger numbers, and is significantly faster in an extension field of degree 2 than in a prime field.

Efficiency of field arithmetic in \mathbb{F}_p and \mathbb{F}_{p^2}

With Schirokauer’s NFS being taken into consideration, the size of our Solinas prime p must be increased to maintain security. We make the assumption that a low weight characteristic field of size $b_{p^k} + \gamma$ provides the same (or greater) security as a general field of size b_{p^k} , and revisit some of the examples from the previous section where security was affected². Then we compare the time required for modular multiplication

²This is justified by the fact that the running time of algorithms to solve discrete logarithms is subexponential.

at the “corrected” bitlevel $b_{p^k} + \gamma$ using Solinas reduction to the time required by general methods at b_{p^k} (the general method chosen was the faster of the REDC and GMP baselines).

#	b_{p^k}	$b_{p^k} + \gamma$	b_{p^k} MRW15	$b_{p^k} + \gamma$ MRW15	b_{p^k} Standard
1	8376	9282	1.5304	1.71002	4.1442
2	15474	17768	3.7852	4.8965	13.457
4	3162	4074	0.1089	0.1650	0.3235
6	15492	21214	1.36799	2.2203	6.0089

Table 6.4: Comparison of modular multiplication times. In examples 4 and 6 the times shown are for multiplication mod a $b_{p^2}/2$ bit prime. We show the timings for 10^4 modular multiplications.

Table 6.4 suggests that the benefits of Solinas arithmetic outweigh the improvements to the NFS. It shows the original field size (b_{p^k}), the number of bits in the enlarged field ($b_{p^k} + \gamma$), then the cost of modular multiplication using GM primes at both of these sizes. The last column shows the time for arithmetic in the original field using a general algorithm, such as Montgomery multiplication. Even after the field size has been increased to maintain the difficulty of the DLP, Solinas reduction provides significant improvement. We used the modular reduction weight of 15 for comparison, however the numbers would differ only slightly for larger MRWs, as the results from Chapter 4 have shown. Recall that this conclusion relies on the assumptions that the field of size $b_{p^k} + \gamma$ provides as much security as the field b_{p^k} . Also, this conclusion is only applicable in general if the improvements to the NFS in these examples is representative of all low weight primes. Until more is done to understand the performance of the modified NFS, hard conclusions will not be possible.

6.3 Recommendations

Collecting up the analysis of this section, we can make some recommendations for pairing-based cryptography.

The embedding degree $k = 1$, in both the supersingular and nonsupersingular cases will unlikely be a viable option where performance is a concern. We make this recommendation based on the difficulty of the finding suitable parameters (as we saw in Chapter 5), and the unfavorable relative performance, even in the case when

Solinas arithmetic is used.

If there is another reason that makes $k = 1$ attractive for a particular application or protocol, then a new curve construction technique should be sought. Since it has the largest operands, \mathbb{F}_p at high security levels can gain the most from Solinas reduction. In addition, low weight prime fields \mathbb{F}_p are less vulnerable to Schirokauer's NFS (see Table 6.3).

We have also strengthened the analysis of [41] for the case of nonsupersingular curves with embedding degree $k \geq 2$. By adding the timings from Chapter 4, we can confirm that, in general higher embedding degrees are better. At the same time we have uncovered some cases where this may not hold, raising doubt that this is always true. We have also confirmed the suggestion that at security levels greater than 192-bits, the Weil pairing gives better performance than the Tate pairing. This recommendation is specific to $k = 2$ (ss and ns). For ordinary curves of larger embedding degree, the work of Granger *et al.* should be considered.

The last recommendation is that the use of Solinas primes should not be abandoned because of the improved NFS. We have shown that the benefits of Solinas reduction greatly outweigh the improvements to the NFS, and that better performance can be achieved with a larger Solinas prime than with a smaller general prime.

The parameters at each security level listed in Table 3.1 should be increased to maintain security when Solinas primes are used. Exactly how much they should be increased is yet to be determined. The decreased security was *estimated* by Schirokauer and until the running time of his NFS variant is better understood, we must tread carefully. On this note, we should say again that [63] is a preprint and still very new work, which may be improved or changed as the research community reviews it.

Chapter 7

Conclusion

During the course of this work, we found many related interesting problems. This chapter first gives some concluding remarks then describes them.

The results of this work answer open questions about the use of generalized Mersenne numbers in computations supporting pairing-based cryptography. We have quantified their importance to these computations and determined their range of applicability. Throughout, our attention to real-world implementation issues makes our contributions ready to be applied to current applications, in both industrial and research settings. Though not a focus of this work, the results in the area of field arithmetic and representation of primes are simultaneously applicable to a wider range of problems in computational number theory. The related problems we have discovered in the process will serve to motivate future work.

7.1 Future Work and Open Problems

The following list of problems are believed to be open at the time of writing. The first half are problems that impact a wide range of research that uses modular arithmetic.

1. Can the MRW of a polynomial $p(t)$ be computed (or estimated) quickly from the coefficients of $p(t)$, without computing the matrix X ?
2. Establish bounds on $\text{MRW}(p)$ for an arbitrary prime p . Then, for a given prime p , what is the lowest MRW representation possible? Is it low enough to allow the efficient reduction methods of §4.3 to be used on *all* primes? If not, what is the density of primes with MRW low enough to be practical? How does this density change when we apply the further restriction that the prime must be word aligned? (as defined in §5.2).
3. Give an algorithm that takes an arbitrary prime p and returns a polynomial

$p(t) \in \mathbb{Z}[t]$ such that $p = p(2^k)$ and $\text{MRW}(p(t))$ is minimal. By minimal, we mean there exists no $q(t) \in \mathbb{Z}[t]$ such that $p = q(2^k)$ **and** $\text{MRW}(q(t)) < \text{MRW}(p(t))$.

4. Generalize the work of [6], which counts the number of GM primes of a certain form, to include a broader families of GM numbers. If possible, restrict the set to those with low modular reduction weight (c.f. Problem 2).
5. Find other families of GM numbers with known MRW, like the one described in §4.6, and those given by Solinas in [74].

This portion of the list deals with problems related to pairing-based cryptography.

6. The modified NFS relies on low-weight primes to find discrete logs quickly. Fast modular arithmetic depends on the MRW but not the weight. Can primes with high weight and low MRW be found? These primes would provide some additional security, however the runtime of Schirokauer's NFS is only weakly dependent on the weight, there is also another property (see [63], page 17). Is it possible to choose primes that sidestep the optimizations of the NFS and simultaneously have low MRW?
7. Determine the practicality of the new NFS, and provide implementation evidence to support or refute the current estimates. Also, describe the performance as the embedding degree rises. Do larger embedding degrees provide more, less, or equivalent security?
8. Devise a curve construction technique to create supersingular curves of embedding degree one where n has low weight and p has low MRW.
9. What is the performance of OEF arithmetic at high security levels? Is it a viable option? Pairing-based cryptography has considered fields of small characteristic (2,3) and large characteristic (as in this work) but not of medium characteristic, i.e. $p < 2^w$ (small enough to fit in a machine word).
10. In [21], Chung and Hasan generalize the notion of GM number further to $m = q^{a_0} \pm q^{a_1} \pm \dots \pm 1$ for *any* prime q . Explore applications of this concept to the problems in the previous list, and also to pairing-based cryptography. Does inclusion of these primes facilitate curve construction and parameter selection?

Appendix A

List of Notation

<i>Symbol</i>	<i>Description, Page number of first appearance</i>
K	a field, 7
$K[x_1, \dots, x_n]$	polynomials in x_1, \dots, x_n with coefficients from K , 7
$GF(p^m)$ or \mathbb{F}_{p^m}	the Galois field with p^m elements, 7
$\exp(G)$	the exponent of a group G , 7
E/K	elliptic curve E over K , 9
$\text{char}(K)$	characteristic of K , 9
\bar{K}	the algebraic closure of a field K , 9
$E(K)$	group of points on the elliptic curve E , 10
\mathcal{O}	the point at infinity, 10
$[m]P$	elliptic curve multiplication, 10
$E(K)[m], E[m]$	m -torsion subgroup, 12
$\#E(K)$	order of the curve group $E(K)$, 12
t	trace of an elliptic curve, 12
$K[E]$	the coordinate ring of E/K , 13
$K(E)$	function field of E/K , 13
u_P	uniformizer at P , 14
$\text{ord}_P(f)$	order of the rational function f at the point P , 14
\mathcal{D}	a divisor, 14
$\text{deg}(\mathcal{D})$	the degree of a divisor, 14
$\text{div}(f)$	the divisor of the rational function f , 14
$\text{Div}(E)$	the group of divisors of E , 14
$\text{Prin}(E)$	the principal divisors of E , 14
$\text{Div}^0(E)$	degree zero divisors of E , 14
$\text{Prin}^0(E)$	degree zero principal divisors of E , 14
$\text{Pic}^0(E) = J(E)$	the Jacobian of E , 14

$e(\cdot, \cdot)$	an abstract pairing, 16
μ_n	the n -th roots of unity, 16
$e_n(\cdot, \cdot)$	the Weil pairing, 16
K^*	the multiplicative group of a field K , 17
$f_{n,P}$	a rational function in $K(E)$ with divisor $n(P) - n(\mathcal{O})$, 17
$\langle \cdot, \cdot \rangle$	the Tate pairing, 18
$L_{P,Q}$	the normalized line through points P, Q , 58
$V_{P,Q}$	the vertical line used to add points P, Q , 58
X	matrix used to determine Solinas reduction function, 39
b_p	the number of bits in p , 75

Appendix B

Test Platform Hardware and Software Particulars

This system was purchased with funds from CFI.

Software Information

uname output:	Linux euclid 2.6.13-gentoo #7 SMP i686 Intel(R) Pentium(R) 4 CPU 3.00GHz GNU/Linux
compiler version:	gcc version 3.4.5 (Gentoo 3.4.5, ssp-3.4.5-1.0, pie-8.7.9)
compiler flags:	Code for field arithmetic: -O2, code for pairings: -O3 -march=i686
GMP version:	4.1.4-r1
GMP-ECM version:	6.1-beta default compiler flags with redc-asm option

Hardware Information

Make and model	Dell Optiplex GX280 (mini-tower)
Microprocessor type	Intel Pentium 4
Level 1 (L1) cache	32 KB
Level 2 (L2) cache	1 MB pipelined-burst, eight-way set associative, write-back SRAM
Memory Type	533 MHz DDR2 SDRAM
Chipset	Intel Grantsdale

Bibliography

- [1] The ECMNET project. <http://www.loria.fr/~zimmerma/ecmnet/>, 2005. Accessed January 2006.
- [2] GMP: The GNU Multiple Precision library. <http://www.swox.com/gmp>, 2005. Accessed January 2006.
- [3] PBC library, 2006. <http://rooster.stanford.edu/~ben/pbc/>, Accessed January 2006.
- [4] L. Adleman. The function field sieve. In *ANTS-I: Proceedings of the First International Symposium on Algorithmic Number Theory*, pages 108–121, London, UK, 1994. Springer-Verlag.
- [5] L. Adleman and M. Huang. Function field sieve method for discrete logarithms over finite fields. *Information and Computation*, 151(1-2):5–16, 1999.
- [6] J. Angel Angel and G. Morales-Luna. Counting prime numbers with short binary signed representation, 2006. Available from <http://eprint.iacr.org/2006/121>.
- [7] D. Bailey. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *Journal of Cryptology*, 14(3):153–176, April 2001.
- [8] D. Bailey and C. Paar. Optimal extension fields for fast arithmetic in public-key algorithms. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, pages 472–485, London, UK, 1998. Springer-Verlag.
- [9] J. Bajard, L. Imbert, and T. Plantard. Modular number systems: Beyond the Mersenne family. In *Lecture Notes in Computer Science, Selected Areas in Cryptography: 11th International Workshop, SAC 2004*, volume 3357, pages 159–170, Berlin, 2004. Springer.
- [10] R. Balasubramanian and N. Koblitz. The improbability that an elliptic curve has subexponential discrete log problem under the MOV algorithm. *J. Cryptology*, 11(2):141–145, 1998.
- [11] P. Barreto, S. 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>.
- [12] P. Barreto, H. Yong Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Proc. of CRYPTO 2002*, pages 354–368, London, UK, 2002. Springer-Verlag.

- [13] P. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees, 2002. <http://citeseer.csail.mit.edu/barreto02constructing.html>.
- [14] P. Barreto, B. Lynn, and M. Scott. Efficient implementation of pairing-based cryptosystems. *J. Cryptology*, 17(4):321–334, 2004.
- [15] P. Barrett. Implementing the Rivest, Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 311–323, London, UK, 1987. Springer-Verlag.
- [16] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *Lecture Notes in Computer Science*, 2139:213, 2001.
- [17] A. Bossalaers, R. Govaerts, and J. Vandewalle. Comparison of three modular reduction functions. In *Advances in Cryptology CRYPTO 1993, Lecture Notes in Computer Science 773, D. R. Stinson (ed.)*, pages 175–186, London, UK, 1993. Springer-Verlag.
- [18] M. Brown, D. Hankerson, J. Lpez, and A. Menezes. Software implementation of the NIST elliptic curves over prime fields. In *Lecture Notes in Computer Science, CT-RSA 2001: The Cryptographers' Track at RSA Conference*, volume 2020, page 250, Berlin, 2001. Springer.
- [19] L. Charlap and R. Coley. An elementary introduction to elliptic curves ii. Technical Report 34, Center for Communications Research - Princeton, July 1990. available from <http://www.idaccr.org/reports/reports.html>.
- [20] L. Charlap and D. Robbins. An elementary introduction to elliptic curves, part I — elliptic curves over algebraically closed fields. Technical Report 31, Center for Communications Research - Princeton, December 1988. Available from <http://www.idaccr.org/reports/reports.html>.
- [21] J. Chung and A. Hasan. More generalized Mersenne primes. Technical Report CORR 2003-17, University of Waterloo, 2003. Also published in SAC 2003.
- [22] D. Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Transactions on Information Theory*, 30(4):587–594, Jul 1984.
- [23] R. Crandall. Method and apparatus for public key exchange in a cryptographic system. United States Patent Number 5159632, Oct 1992.
- [24] R. Crandall and B. Fagin. Discrete weighted transforms and large-integer arithmetic. *Math. Comput.*, 62(205):305–324, 1994.
- [25] R. Crandall and C. Pomerance. *Prime Numbers: A Computational Perspective*. Springer-Verlag, New York, 2002.

- [26] J. Daemen and V. Rijmen. *The Design of Rijndael: AES — The Advanced Encryption Standard*. Springer, New York, 2002.
- [27] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.
- [28] I. Blake (Editor). *Elliptic Curves in Cryptography*. Number 265 in London Mathematical Society Lecture Note Series. Cambridge University Press, 1999.
- [29] I. Blake (Editor). *Advances in Elliptic Curve Cryptography*. Number 317 in London Mathematical Society Lecture Note Series. Cambridge University Press, 2005.
- [30] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology: Proceedings of CRYPTO 84*, pages 10–18. Springer-Verlag, 1985.
- [31] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
- [32] J. Fraleigh. *A First Course in Abstract Algebra*. Addison Wesley, Boston, seventh edition, 2003.
- [33] G. Frey, M. Muller, and H. Ruck. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Transactions on Information Theory*, 45(5):1717–1719, 1999.
- [34] S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In *ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory*, pages 324–337, London, UK, 2002. Springer-Verlag.
- [35] D. Gordon. Discrete logarithms in $gf(p)$ via the number field sieve. *SIAM J. Discrete Math.*, 16:124–138, 1993.
- [36] R. Granger, D. Page, and N. Smart. High security pairing-based cryptography revisited. Available from: <http://eprint.iacr.org/2006/059>.
- [37] T. Granlund. *The GNU Multiple Precision Arithmetic Library, Edition 4.1.4*. Available from: <http://www.swox.com/gmp/manual/>.
- [38] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.
- [39] D. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, second edition, 1981.
- [40] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, Jan 1987.

- [41] N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. Cryptology ePrint Archive, Report 2005/076, 2005. <http://eprint.iacr.org/>.
- [42] K. Koc, T. Acar, and B. Kaliski. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–32, 1996.
- [43] A. Lenstra. Unbelievable security. matching AES security using public key systems. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 67–86, London, UK, 2001. Springer-Verlag.
- [44] A. Lenstra and E. Verheul. Key improvements to XTR. In *ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*, pages 220–233, London, UK, 2000. Springer-Verlag.
- [45] A. Lenstra and E. Verheul. The XTR public key system. In *CRYPTO '00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, pages 1–19, London, UK, 2000. Springer-Verlag.
- [46] A. Lenstra and E. Verheul. Fast irreducibility and subgroup membership testing in XTR. In *PKC '01: Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, pages 73–86, London, UK, 2001. Springer-Verlag.
- [47] A. Lenstra and E. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, December 2001.
- [48] U. Maurer and S. Wolf. Diffie-Hellman oracles. In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 268–282, London, UK, 1996. Springer-Verlag.
- [49] A. Menezes, T. Okamoto, and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. In *STOC '91: Proc. of the twenty-third annual ACM symp. on theory of computing*, pages 80–89, 1991.
- [50] A. Menezes, P van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. Available online at <http://www.cacr.math.uwaterloo.ca/hac/>.
- [51] V. Miller. Short program for functions on curves, 1986. Unpublished manuscript, available from <http://crypto.stanford.edu/miller/miller.pdf>.
- [52] V. Miller. Use of elliptic curves in cryptography. In *Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc.
- [53] V. Miller. The Weil pairing, and its efficient calculation. *J. Cryptology*, 17(4):235–261, 2004.

- [54] P. Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–523, 1985.
- [55] A. Odlyzko. Discrete logarithms: The past and the future. *Designs, Codes and Cryptography*, 19(2-3):129–145, March 2000.
- [56] National Institute of Standards and Technology (NIST). Digital signature standard (dss). *FIPS186-2: Federal Information Processing Standards Publication 186*, Jan 2000. Available from : <http://csrc.nist.gov/publications/fips/index.html>.
- [57] National Institute of Standards and Technology (NIST). Announcing the advanced encryption standard (AES). *FIPS197: Federal Information Processing Standards Publication 197*, Nov 2001. Available from : <http://csrc.nist.gov/CryptoToolkit/aes/rijndael/>.
- [58] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 24:106–110, January 1978.
- [59] J. Pollard. Monte Carlo methods for index computation mod p . *Mathematics of Computation*, 32(143):918, 1978.
- [60] J. Pollard. Kangaroos, Monopoly and discrete logarithms. *Journal of Cryptology*, 13:437–447, 2000.
- [61] T. Satoh and K. Araki. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. *Commentarii Math. Univ. St. Pauli*, 47:81–92, 1998.
- [62] O. Schirokauer. The special function field sieve. *SIAM Journal on Discrete Mathematics*, 16(1):81–98, 2002.
- [63] O. Schirokauer. The number field sieve for integers of low weight, 2006. Preprint. Available from: <http://eprint.iacr.org>.
- [64] O. Schirokauer, D. Weber, and T. Denny. Discrete logarithms: the effectiveness of the index calculus method. In *Algorithmic Number Theory: Proc. ANTS II, Talence, France, volume 1122 of Lecture Notes in Computer Science*, pages 337–361. Springer-Verlag, 1996.
- [65] I. Semanev. Evaluation of discrete logarithms on some elliptic curves. *Mathematics of Computation*, 67:353–356, 1998.
- [66] A. Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

- [67] D. Shanks. Class number, a theory of factorization, and genera. In *Proceedings of the Pure Math Symposium*, pages 415–440. AMS, Providence, R.I., 1971.
- [68] V. Shoup. Lower bounds for discrete logarithms and related problems. *Advances in Cryptology, EUROCRYPT 97*, pages 313–328, 1997.
- [69] V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, <http://shoup.net/ntb/>, 2005. Available Online.
- [70] J. Silverman. *The arithmetic of elliptic curves*. Graduate Texts in Mathematics 106. Springer-Verlag, New York, 1986.
- [71] J. Silverman and J. Tate. *Rational Points on Elliptic Curves*. Springer-Verlag, New York, 1992.
- [72] N. Smart. Elliptic curve cryptosystems over small fields of odd characteristic. *Journal of Cryptology*, 12(2):141–151, 1999.
- [73] N. Smart. A comparison of different finite fields for elliptic curve cryptosystems. *Computers & Mathematics with Applications*, 42(1):91–100, July 2001.
- [74] J. Solinas. Generalized Mersenne primes. Technical Report CORR 99-39, University of Waterloo, 1999.
- [75] E. Teske. Square-root algorithms for the discrete logarithm problem. (a survey). Technical Report CORR 98-52, University of Waterloo, 1998.
- [76] P. van Oorschot and M. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In *CCS '94: Proceedings of the 2nd ACM Conference on Computer and communications security*, pages 210–218, New York, NY, USA, 1994. ACM Press.
- [77] E. Verheul. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. *J. Cryptology*, 17(4):277–296, 2004.
- [78] Erik De Win, Serge Mister, Bart Preneel, and Michael J. Wiener. On the performance of signature schemes based on elliptic curves. In *ANTS-III: Proceedings of the Third International Symposium on Algorithmic Number Theory*, pages 252–266, London, UK, 1998. Springer-Verlag.