On Small Degree Extension Fields in Cryptology

Robert Granger

A dissertation submitted to the University of Bristol in accordance with the requirements for the degree of Doctor of Philosophy in the Faculty of Engineering, Department of Computer Science.

50,000 words

November 2005

Abstract

This thesis studies the implications of using public key cryptographic primitives that are based in, or map to, the multiplicative group of finite fields with small extension degree. A central observation is that the multiplicative group of extension fields essentially decomposes as a product of algebraic tori, whose properties allow for improved communication efficiency.

Part I of this thesis is concerned with the constructive implications of this idea. Firstly, algorithms are developed for the efficient implementation of torus-based cryptosystems and their performance compared with previous work. It is then shown how to apply these methods to operations required in low characteristic pairing-based cryptography. Finally, practical schemes for high-dimensional tori are discussed. Highly optimised implementations and benchmark timings are provided for each of these systems.

Part II addresses the security of the schemes presented in Part I, i.e., the hardness of the discrete logarithm problem. Firstly, an heuristic analysis of the effectiveness of the Function Field Sieve in small characteristic is given. Next presented is an implementation of this algorithm for characteristic three fields used in pairing-based cryptography. Finally, a new index calculus algorithm for solving the discrete logarithm problem on algebraic tori is described and analysed.

Acknowledgements

It has been just over three years since I began studying for this degree. These three years have been a tremendous period of development for me, both academically and personally, and I owe a sincere debt of gratitude to many people who have made much of this time unforgettable.

First and foremost, I would like to thank my supervisor Nigel Smart. His seemingly boundless knowledge, enthusiasm and sense of humour have been an inspiration, and without his expert guidance I would no doubt not be writing this.

I would also like to extend my thanks to Fré Vercauteren, who as my advisor for two years was always available to bounce ideas off (and indeed still is), and often bounced several new ones back to me with an infectious alacrity.

My learning during the course of this PhD has by no means been a lone effort. The Cryptography and Information Security Group at Bristol has been an excellent place to work, and contains (and has contained) several talented people, who have always been willing to share their expertise, and a joke. Many thanks go to Dan Page, John Malone-Lee, Martijn Stam, Peter Leadbitter, Richard Noad, Florian Hess, Katharina Geissler, Pooya Farshim, Kamel Bentahar, and also to Steven Galbraith, for answering my often ill-posed questions when no one else could. Thanks also to Mark, Dennis, Kate, Angus, Veronica, Paul and Mike, for splitting up the days, and stretching out the nights, and to Amoss also, who as well as being a great mate, first piqued my interest in real computer science.

In addition to going to more places than I can remember, and meeting even more very interesting people, I had the privilege of being invited to visit the CACR at the University of Waterloo for three brilliant months in the summer of 2005.

Very special thanks go to Alfred Menezes for being an excellent host, and also to Edlyn Teske, Darrel Hankerson, Nicholas Thériault and Isabelle Dechene for making my stay both stimulating and a real pleasure. A very big thanks also to Kim, Erin and Iza, for taking me under their wing, and showing me just how damn awesome life can be.

Thanks also to my friends who have always been up for a good laugh over the years: to Russ, Mozza, Joey, Bourner, Lovell, Pete and Sutton - cheers lads; to Greg, for opening my eyes and for suggesting doing this PhD in the first place; to Kuba, for being a really good mate; to Michal and Ania, for throwing wicked parties; and to Phil, who over the last three years not only taught me what to look for, but also always made sure I kept my head firmly fixed on.

And finally, a huge thanks to my family: to Mum and Dad - thank you for all your support over the years; to Amy and James, for being there no matter what; to Alex, for making me feel old(er); and to Jasmine and Tamara, who I have not seen enough of over the last seven years - now it's time to make sure it was worth it...

Declaration

I declare that the work in this dissertation was carried out in accordance with the Regulations of the University of Bristol. The work is original except where indicated by special reference in the text and no part of the dissertation has been submitted for any other degree.

Any views expressed in the dissertation are those of the author and in no way represent those of the University of Bristol.

The dissertation has not been presented to any other University for examination either in the United Kingdom or overseas.

SIGNED: DATE:

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

"It is insufficient to protect ourselves with laws; we need to protect ourselves with mathematics."

– Bruce Schneier, security technologist and author.

"There is no safety in numbers, or in anything else."

– James Thurber, author, cartoonist, humorist, & satirist.

In this chapter we provide a synopsis of modern cryptology, detail the focus and overall structure of the thesis, and describe the main results contained herein.

1.1 Cryptology

Cryptology is the science of making and breaking secret communications. This description, while perhaps too general to convey the sophistication of modern cryptology, embodies the subject's central dichotomy. Should one party for instance, design and implement a method to communicate a message secretly in the presence of an adversary (cryptography), then assuming the message is sufficiently valuable, this adversary is likely to attempt to circumvent the design (cryptanalysis). The resulting 'arms race' between cryptographer and cryptanalyst, with regard to both the problem of confidential data transmission and the manifold applications of modern cryptography, constitutes the field of cryptology.

Historically, the development and deployment of cryptography has been driven by military and governmental practices of strategic or political importance. However in recent times, the prevalence of cryptography in modern society has increased dramatically. With the widespread availability of cheap computer technology, the rapid growth of the internet, and the increasing popularity of wireless communications and electronic commerce, numerous difficult problems in information security have arisen. The variety and depth of these problems has stimulated intense research activity in the past three decades, particularly in academia and industry, which in turn has resulted in a revolution in cryptographic techniques and methodology.

Whereas, in the past, the design of cryptosystems has tended to be ad hoc, modern cryptography can now claim to be a scientific discipline (though one that is still maturing). A central feature of this maturity is the trend towards modularisation. By identifying basic security objectives and finding mathematical objects which satisfy these objectives, cryptographers are now able to design, compose, and argue the security of complex cryptosystems in a rigorous manner.

The most basic of the modules in modern cryptography is that of a *primitive*, which may be regarded as a cryptographic building block which performs one or more desired functions, and may be combined with others to form a cryptographic *protocol*. The most well-known and perhaps the simplest primitive is *encryption*, which allows parties to achieve confidential data transmission over an insecure channel. The concept of a *digital signature* is another essential primitive, enabling parties to perform data-origin or entity authentication, while *key agreement* allows parties to securely establish a common key over an insecure channel, for use in encryption for example. In general, primitives are designed to satisfy particular security objectives, which may be built from the following four basic objectives [101]: *confidentiality, data integrity, authentication* and *non-repudiation*.

By isolating the basic properties required of a given cryptosystem in this way, and developing primitives that satisfy these objectives, sound design practices can be followed, which in turn permit simpler security analyses.

The process of combining cryptographic primitives to achieve a particular functionality and to satisfy certain security objectives is intricate, and to some extent depends on what one can argue regarding the security obtained by a given design. Ideally, one provides a reductionist security argument, reducing the security of the protocol to the security of the underlying primitives. The techniques employed in such arguments constitute the area of cryptography known as *provable security*. Informally, one argues the security of the protocol by showing that if an adversary is capable of breaking the protocol (according to some definition of security) with non-negligible probability, then this adversary would be able to use this information to break one (or more) of the underlying primitives with non-negligible probability as well. If one then assumes that the underlying primitives are secure, one deduces that such an adversary can not exist. Such a proof then demonstrates the conceptual soundness of the protocol design.

Given a protocol design with a corresponding proof of security relative to a set of cryptographic primitives, the next task in the design process is to find the technical means by which these primitives may be securely instantiated. The methods employed during this stage draw from fields as diverse as computational complexity theory, information theory, discrete mathematics, computational number theory, computational algebraic geometry, quantum computation and quantum information theory, to name a few. The number of possible methods of instantiating a given primitive is huge, but there is no *a priori* guarantee that any of these ideas are secure (except possibly for quantum key distribution [31], which potentially holds great promise for the future if it can be made practical - see [108]). Indeed, despite many years of research, no modern instantiations of any non-quantum primitives have been proven secure either in absolute or complexity theoretic terms, and indeed may never be [115]. As such, one must rely on the continuing failure of cryptanalyst's best efforts to break these primitives, using this as an assurance that the problems are as hard as believed, where here "hard" means the expected time taken to solve a given problem instance is perhaps millions, if not billions, of years.

With the difficulty of these problems established, or at the very least assumed but with strong supporting evidence, to be of any use a protocol must be implemented in the real world. This raises several separate security issues, since physical computation potentially (and nearly always) leaks information regarding the secret data being operated upon (see chapter 4 of [11] for some of the methods employed). Hence further defences, in the form of countermeasures, must be developed.

These issues having been addressed and the system deployed, one must still continually reassess the security level attained by a given protocol, in response to both specialised cryptanalytic attacks, and in light of developments in any of the above technical fields or in computer technology. As such, contemporary cryptology is a fast-moving subject requiring the solving of serious and difficult problems, using a plethora of mathematical, scientific and engineering techniques.

1.2 Symmetric Key Cryptography

While not the subject of this thesis, for completeness we briefly describe symmetric key cryptography, which is widely used today in situations where shared keys amongst parties have already been established. The Data Encryption Standard (DES) [37,38] and its replacement, the Advanced Encryption Standard (AES) [26, 40] are the most well known encryption schemes of this type.

Let $\mathcal{M}, \mathcal{C}$ and $\mathcal{K}$ be the message, ciphertext and key spaces respectively for the following scenario. Suppose two parties A and B wish to communicate over an insecure channel, and assume that they have agreed upon a family of encryption functions (indexed by $k \in \mathcal{K}$)

$$E_k : \mathcal{M} \rightarrow \mathcal{C},$$

with a family of corresponding decryption functions

$$D_k : \mathcal{C} \rightarrow \mathcal{M},$$

in the sense that for each $k \in \mathcal{K}$ and every $m \in \mathcal{M}$, $D_k(E_k(m)) = m$. If A and B secretly agree a shared key k, then A can confidentially transmit a message m to B as follows. First A computes the ciphertext $c = E_k(m)$, and sends this to B. To decrypt, B simply computes $D_k(c) = m$, recovering the original message. Note that any unauthorised party E can read c, since the channel is insecure, but without the key k, E should not be able to obtain any information regarding m. Also note that the roles of A and B in this scenario can be reversed, so that B can confidentially transmit a message to A using the same secret key k.

In actual systems, the decryption key may not be identical to the encryption key, but it is always easily derivable from it (in polynomial time), and so both encryption and decryption can be performed with the knowledge of a single key, thus explaining the terminology.

While symmetric key based primitives can provide confidentiality, and amongst the parties sharing a key, message authentication, one problem is the establishment of pairwise secret keys. Thus when the number of communicating parties is large, as has increasingly been the case since the advent of the internet, this becomes a serious logistical problem, requiring a totally new idea to overcome it.

1.3 Public Key Cryptography

Up until the mid 1970's, symmetric key based cryptographic techniques were the only ones available. The major limitation of these methods was the key distribution problem: how should a large number of parties efficiently establish pairwise secure keys, if they are potentially all over the world? In 1976 the revolutionary paper of Diffie and Hellman [28] gave a method to solve this problem, with the introduction of the notion of public key cryptography.

Rather than each pair of parties sharing a secret key, each party I now has a pair of keys (e_I, d_I), where e_I is published (the public key), and d_I is secret. In order for party A to securely send a message m to party B, using B's public key e_B, A computes $c = E_{e_B}(m)$ and sends c to B. Since d_B is assumed to be known only to B, only B can apply the inverse transformation D_{d_B} to c, thus recovering

m. Hence there is no need for parties to share keys at all, and the key distribution problem is bypassed.

The essential assumption in this set-up, and what makes it substantially different to symmetric cryptography, is that it should be computationally infeasible to determine d_I from e_I. This is why public key cryptography is also often called *asymmetric cryptography*.

Another benefit of public key encryption is that if a system is reversible, meaning that decryption can be performed before encryption (which implicitly assumes that the message and ciphertext are the same), then one also naturally obtains digital signatures which provide both authentication and non-repudiation (see Sections 1.3.1 and 1.3.2 for examples).

Hence public key cryptography seems to provide an excellent solution not only to the key distribution problem, but also to the issues information security sets out to resolve. One disadvantage however is that asymmetric cryptosystems are usually several orders of magnitude slower than their symmetric counterparts, while another more technical problem is that in using public keys, the keys themselves must first be authenticated. The latter can be overcome with an appropriate *public key infrastructure* [145], while the former is usually circumvented by combining both types of system: one uses asymmetric cryptography for key establishment and then symmetric cryptography for secure and efficient data transfer.

1.3.1 RSA

With the concept of public key cryptography having been established, in 1977 the first concrete instantiation was proposed by Rivest, Shamir and Adleman [124], called RSA, after its inventors. In RSA a public key consists of a pair (n, e). Here n is a large integer which is usually chosen to be the product of two primes of approximately equal size, and e is called the encryption exponent, which is an integer in $(\mathbb{Z}/\phi(n)\mathbb{Z})^\times$. The corresponding secret key d is chosen such that $ed \equiv 1 \bmod \phi(n)$. For party A to encrypt a message $m \in \mathbb{Z}/n\mathbb{Z}$ to party B, he computes $c = m^{e_B} \bmod n_B$, from which B recovers m as $c^{d_B} \equiv m^{e_B d_B} \equiv m \bmod n_B$. The security of RSA encryption relies on both the difficulty of integer factorisation and the difficulty of computing e-th roots modulo a composite for which the prime factorisation is not known, which for security analyses are generally presumed to be equivalent. However, remarkable work by Boneh and Venkatesan [16] implies that for small encryption exponents, the two problems are equivalent only if factorisation is easy, and so it is likely that the two problems are in general not equivalent.

With a suitably chosen key-size, padding scheme, implementation, and with proper usage, RSA is believed to be secure. In fact the RSA scheme was independently designed some four years previously at GCHQ by Cocks [22], building upon an idea of Ellis [33]. At the time GCHQ insisted on keeping the system secret, and did not use it internally, purportedly due to the fact that they were not at the time able to obtain digital signatures (and hence certificates), which in retrospect now seems very easy [28]. Indeed, to sign a message m party A merely publishes m and $s = m^{d_A} \bmod n_A$, from which anyone can verify whether or not A signed the message by checking if $s^{e_A} = m^{d_A \cdot e_A} = m \bmod n_A$, since e_A is public. Note that in this scheme, A is really only providing a proof of knowledge of d_A, which it is presumed only they know, hence uniquely identifying A as the signer.

Since the difficulty of integer factorisation is not proven, and in fact may never be [115], it is sensible to have more than one instantiation of public key cryptography.

1.3.2 Discrete Logarithm Cryptography

Currently the only widespread public key crytptosystems other than RSA are based on the discrete logarithm problem (DLP). For an arbitrary finite cyclic group G, written multiplicatively, and for a fixed generator $g \in G$ and an element $h \in G$, the discrete logarithm problem with respect to g and h is to determine an integer x which satisfies the equation

$$g^x = h.$$

Assuming this problem is hard, one can instantiate key agreement, encryption and digital signatures.

First proposed by Diffie and Hellman [28] using the multiplicative group of finite fields, the security of the following key agreement protocol is related to the difficulty of solving discrete logarithms. If two parties A and B wish to agree upon a common key over an insecure channel, they first publicly agree upon a finite cyclic abelian group G, and a fixed (generating) element $g \in G$. A then chooses a random integer $a \bmod \#G$ and sends g^a to B. Similarly B chooses a random integer $b \bmod \#G$ and sends g^b to A. With these elements exchanged, A then computes $(g^b)^a$, and B computes $(g^a)^b$, which by the commutativity of exponentiation provides a common key g^{ab}. An eavesdropper E only obtains G, g, g^a and g^b, and if she can recover from this data g^{ab}, she is said to have solved the *Diffie-Hellman problem* (DHP). It is easy to see that if E can solve discrete logarithms in G, then she can break the protocol and can solve the DHP. For most groups in use in cryptography it is believed that the DHP and DLP are equivalent, though this has been proven for a restricted class of groups only [97].

It was not until 1985 that ElGamal proposed encryption and digital signature schemes based on the DLP [32]. Encryption is achieved as follows. Suppose B has a public key consisting of g and $h = g^x$, where x is the private key. To encrypt a message m, assumed to be encoded as an element of the group G, A generates a random integer $k \in \{1, \dots, \#G - 1\}$ and computes $a = g^k$, $b = h^k m$, and sends (a, b) to B. The message is then recovered by B who computes $ba^{-x} = h^k m g^{-kx} = g^{xk-xk} m = m$.

With regard to digital signatures, suppose B wants to sign a message $m \in (\mathbb{Z}/(\#G)\mathbb{Z})$. We fix a bijection $f : G \to \mathbb{Z}/(\#G)\mathbb{Z}$. Then using the same public and private key pair as for encryption, B generates a random integer $k \in \{1, \dots, \#G - 1\}$, and computes $a = g^k$. He then computes a solution, $b \in \mathbb{Z}/(\#G)\mathbb{Z}$, to the congruence $m \equiv xf(a) + bk \pmod{\#G}$, and sends the signature (a, b) and m to A. To verify the signature A checks whether the following equation holds:

$$h^{f(a)} a^b = g^{xf(a)+kb} = g^m.$$

Another example is the Digital Signature Algorithm (DSA), used in the Digital Signature Standard (DSS), first proposed and standardised by the U.S. government's National Institute of Standards and Technology in 1994, and revised most recently in 2000 [39]. The DSA is almost identical to ElGamal signatures, except that the verification procedure is computationally simpler.

1.3.3 Elliptic Curve Cryptography

The security of the schemes discussed in the previous section is related to the difficulty of the DLP in a given group. However from a representation perspective using the multiplicative group of a finite field, as suggested by Diffie and Hellman, is not optimal. The reason for this is that current index calculus algorithms can solve discrete logarithms in finite fields approximately six times the size of groups to which generic algorithms apply [3, 52, 70, 142, 151]. As a consequence this representation is about six times less efficient in terms of memory and bandwidth than the information-theoretic limit. In the future this ratio will continue to increase as recommended key sizes become larger, since algorithms to solve finite field discrete logarithms are subexponential.

Inspired by H.W. Lenstra's elliptic curve integer-factorisation algorithm [90], elliptic curve cryptography (ECC), proposed independently by Miller [103] and Koblitz [74] in 1985, overcomes this inefficiency. Besides a few easily identifiable cases, solving the DLP in the group of rational points on an elliptic curve over a finite field is believed to be a hard problem, for which the best known-algorithms are in general exponential. This means that cryptosystems based on elliptic curves are essentially optimal [142]. Relative to systems based on the DLP in the multiplicative group, one can therefore use much shorter keys and hence these systems require less bandwidth, less power consumption and less silicon area, making elliptic curve based systems ideal in constrained environments such as smart cards, for which such commodities are at a premium.

Elliptic curve cryptography has been thoroughly researched for the last twenty years, and despite some recent advances in the elliptic curve DLP (ECDLP) for curves defined over some extension fields [27, 47, 48], confidence in the difficulty of the underlying computational problem remains high, with many modern cryptosystems relying on the assumed intractability of the ECDLP. This confidence was recently given a huge boost with the announcement by the NSA that they will now be using ECC for both digital signatures and key exchange [112].

1.3.4 Identity-Based Cryptography

The concept of identity-based cryptography (IBC) was originally proposed by Shamir [140] in 1984, in order to reduce the complexity of public key infrastructure systems, which must scale to process large numbers of authentications. The basic idea behind IBC is that by using the notion of identity as a user's public key, the amount of infrastructure required is greatly reduced since a message sender implicitly knows the public key of the recipient. An often used example is that of an email address. Within this context, an identity for party A might be the string

`alice@gmail.com`.

If party B wants to send A secure email, B implicitly knows A's email address and hence their identity and public key. Therefore B can encrypt the email to A without the same level of involvement from, for example, certificate and trust authorities. IBC has thus been an attractive target for researchers since its inception.

One of the most important developments of the last decade was the realisation that what had previously been used as a method of attack on the ECDLP [41,100], could be used to instantiate IBC [14, 129]. The use of non-degenerate bilinear maps, or pairings, most easily computed over elliptic curve groups, has caused a minor revolution in cryptography, and there are now over 200 papers making essential use of their properties (cf. [153]).

One initial drawback of these systems is that a pairing computation was typically ten times slower than a point multiplication on an elliptic curve. Much research has thus focused on the efficient implementation of elliptic curve pairing computation [7, 43]. In many protocols other field operations are required and their efficiency can be improved with some of the techniques available in torus-based cryptography, which we now introduce.

1.3.5 Torus-Based Cryptography

The first instantiation of public key cryptography, the Diffie-Hellman key agreement protocol [28], was based on the assumption that discrete logarithms in finite fields are hard to compute. During the 1980's, this protocol and the signature and encryption schemes due to ElGamal [32], were formulated in the full multiplicative group of a prime finite field $\mathbb{F}_p$. To speed-up exponentiation and obtain shorter signatures, Schnorr [134] proposed to work in a small prime order subgroup of the multiplicative group of $\mathbb{F}_p$. Most modern DLP-based cryptosystems, such as the Digital Signature Algorithm (DSA) [39], follow Schnorr's idea.

In 1995, Smith and Skinner described the cryptosystem LUC [146], which eliminates some of the redundancy inherent in using subgroups of finite fields. By representing elements of the order $p+1$ subgroup of $\mathbb{F}_{p^2}^{\times}$ with their trace, one needs just one element of $\mathbb{F}_p$ to identify an element, rather than two, thus providing compression.

The next exploitation of these subgroups was due to Lenstra [85], who showed that by working in the order $\Phi_n(p)$ cyclotomic subgroup of $\mathbb{F}_{p^m}^{\times}$, for extensions that admit an optimal normal basis, one can obtain improved exponentiation times.

Then in 1999, building upon the idea behind LUC, Brouwer, Pellikaan and Verheul [19] described a system achieving a compression factor of three for elements in the order p^2-p+1 subgroup of $\mathbb{F}_{p^6}^{\times}$. They further conjectured that one can attain a compression ratio of $n/\phi(n)$ for elements of the cyclotomic subgroup of $\mathbb{F}_{p^n}^{\times}$. Later, Lenstra and Verheul developed XTR [88, 89], extending the compression method in [19] to include cryptographic operations.

These considerations may seem superfluous in the presence of ECC, since for ECC the representations are essentially optimal, but there are two basic reasons why this is not the case. Firstly, it is unreasonable to presume that progress in attacks for the ECDLP will not perhaps one day make these systems significantly weaker, so it is prudent to have alternative systems to fall back on. Secondly, in the efficient implementation of identity-based systems, one manipulates elements of fields of small extension degree, and hence it is of interest to have efficient methods to perform these operations, and to reduce the redundancy inherent in this representation.

In 2003, Rubin and Silverberg [127] showed how to interpret and generalise the above cryptosystems, recasting the problem of compression for extension fields in terms of algebraic tori. Torus-based cryptography (TBC) may be regarded as a natural extension of classical Diffie-Hellman and ElGamal in a finite field $\mathbb{F}_p$, where key agreement, encryption and signature schemes are performed in the multiplicative group $\mathbb{F}_p^\times$. For any positive integer n, one can define an algebraic torus T_n over $\mathbb{F}_p$ such that over $\mathbb{F}_{p^n}$, this variety is isomorphic to $\phi(n)$ copies of $\mathbb{F}_p^\times$, where $\phi(n)$ is the dimension of T_n. In fact T_n is nothing other than the cyclotomic subgroup of $\mathbb{F}_{p^n}^\times$ [127]. When T_n is 'rational', it is possible to embed T_n in $\phi(n)$-dimensional affine space, and thus represent every element by just $\phi(n)$ elements of $\mathbb{F}_p$. The exploitation of the rationality of tori provides a compression factor of $n/\phi(n)$ for elements in the cyclotomic subgroup of $\mathbb{F}_{p^n}^\times$, which explains current interest in the area. If n is the product of at most two prime powers then T_n is known to be rational [81, 161]. Based on the rationality of T_6, Rubin and Silverberg [127] developed the CEILIDH public key cryptosystem, which achieves the same compression factor of three as XTR.

Only recently was the connection between algebraic tori and the existing trace-based systems LUC [146] and XTR [85] made explicit [127]. Of particular interest is a current conjecture about algebraic tori that if true, implies the existence of cryptosystems based in $\mathbb{F}_{p^n}^\times$ with arbitrarily large compression ratio. In fact the necessity of this conjecture has been bypassed [156, 157] using alternative, proven properties of tori, and various avenues in this field have yet to be fully explored.

1.4 Thesis Outline and Main Contributions

This thesis studies the implications of using public key cryptographic primitives that are based in, or map to, the multiplicative group of finite fields with small extension degree. A central observation is that the multiplicative group of extension fields essentially decomposes as a product of algebraic tori, whose properties allow for improved communication efficiency.

The motivation for these considerations comes from two related areas. Firstly, such fields are the natural setting for the relatively new area of torus-based cryptography, which until the work detailed in this thesis was commenced, were not studied at all in terms of efficiency. Secondly, identity-based cryptosystems based on pairings over algebraic groups map to algebraic tori, and so all of the techniques available for TBC transfer to the implementation of IBCs. Hence any results regarding the former apply to both, whether they be developments in arithmetic or weaknesses in security.

Broadly, the thesis is divided into two parts. In Part I, we describe algorithms and representations for efficient arithmetic on algebraic tori, and apply these to some identity-based schemes, improving their efficiency. We also describe new efficient systems for high dimensional tori. Highly optimised implementations and benchmark timings are provided for each of these systems.

In Part II we concern ourselves with the computational problem underlying the security of these schemes, namely, the discrete logarithm problem. We first give estimates of the effectiveness of the Function Field Sieve for the computation of discrete logarithms in small characteristic fields. We then describe an implementation of this algorithm for characteristic three fields with composite extension degree. In the final chapter, we present a new algorithm for solving the DLP on algebraic tori, and analyse its performance.

In the next chapter, preceding Part I, we also give some relevant mathematical background. The work contained in this thesis has led to a number of publications [54–57, 59, 156], and also [58], which we omit for reasons of space.

Chapter 2

Mathematical Background

This chapter fixes the notation used throughout the thesis, and briefly recalls the relevant definitions and results required.

2.1 Finite Fields

For p a prime let $\mathbb{F}_p$ denote the prime Galois field consisting of p elements. For an integer $n > 1$ we denote by $\mathbb{F}_{p^n}$ the degree n extension of $\mathbb{F}_p$, or some representation thereof. We will often refer to a generic finite field simply as $\mathbb{F}_q$, where it is understood that q is a prime power p^m, or to a specific degree n extension of a generic finite field as $\mathbb{F}_{q^n}$. When referring to any field $\mathbb{F}_q$, with $q = p^m$, we call the subfield $\mathbb{F}_p$ of $\mathbb{F}_q$ its *prime subfield*. We also write $\mathbb{F}_{q^n}^{\times}$ for the group of units, or multiplicative group of, $\mathbb{F}_{q^n}$, and $\overline{\mathbb{F}}_q$ for an algebraic closure of $\mathbb{F}_q$.

We denote by $\mathrm{Gal}(\mathbb{F}_{q^n}/\mathbb{F}_q)$ the Galois group of the extension $\mathbb{F}_{q^n}/\mathbb{F}_q$, which is cyclic and of order n, and is generated by the Frobenius automorphism

$$\varphi : \left\{ \begin{array}{ccc} \overline{\mathbb{F}}_q & \longrightarrow & \overline{\mathbb{F}}_q, \\ \alpha & \longmapsto & \alpha^q. \end{array} \right.$$

Two maps which arise naturally from the extension $\mathbb{F}_{q^n}/\mathbb{F}_q$, and will be useful to us, are the *trace* and *norm* maps, which we now define.

Definition 2.1. *For $\alpha \in \mathbb{F}_{q^n}$, the* trace $Tr_{\mathbb{F}_{q^n}/\mathbb{F}_q}(\alpha)$ *of α over $\mathbb{F}_q$ is defined to be*

$$Tr_{\mathbb{F}_{q^n}/\mathbb{F}_q}(\alpha) = \alpha + \varphi(\alpha) + \cdots + \varphi^{n-2}(\alpha) + \varphi^{n-1}(\alpha).$$

Definition 2.2. *For $\alpha \in \mathbb{F}_{q^n}$, the* norm $N_{\mathbb{F}_{q^n}/\mathbb{F}_q}(\alpha)$ *of α over $\mathbb{F}_q$ is defined to be*

$$N_{\mathbb{F}_{q^n}/\mathbb{F}_q}(\alpha) = \alpha \cdot \varphi(\alpha) \cdot \cdots \cdot \varphi^{n-1}(\alpha).$$

For elementary properties of the trace and norm, we refer the reader to pp. 51-55 of [92], and for a comprehensive introduction to finite fields in general, we refer the interested reader to the same book, in which much of what follows can be found.

2.1.1 Structure of Extension Fields

As before let $\mathbb{F}_{q^n}$ be a degree n extension of $\mathbb{F}_q$. Then for each positive divisor d of n, $\mathbb{F}_{q^n}$ contains exactly one subfield of order q^d, namely, $\mathbb{F}_{q^d}$, and conversely, every subfield of $\mathbb{F}_{q^n}$ containing $\mathbb{F}_q$ has order q^d, where d is a positive divisor of n (see Theorem 2.6 of [92]).

For a given n, the subfields of $\mathbb{F}_{q^n}$ can therefore be determined by listing all the positive divisors of n, and the containment relations between subfields correspond exactly to the divisibility relations among the positive divisors of n. Subfield membership is characterised by the Frobenius automorphism, so that an element $\alpha \in \mathbb{F}_{q^n}$ is in $\mathbb{F}_{q^d}$ if and only if $\varphi^d(\alpha) = \alpha$. In fact one can define $\mathbb{F}_{q^d}$ to be the set of solutions to this equation, i.e., the splitting field of the polynomial $x^{q^d} - x \in \mathbb{F}_q[x]$.

2.1.2 The Multiplicative Group and Subgroup Embeddings

For certain applications in cryptography, of interest is not the full field $\mathbb{F}_{q^n}$, but its multiplicative group, $\mathbb{F}_{q^n}^{\times}$. As we detail below, the polynomial $x^n - 1 \in \mathbb{F}_q[x]$ factors algebraically, leading to a simple classification for subfield membership of subgroups of $\mathbb{F}_{q^n}^{\times}$. We first require some background on roots of unity and cyclotomic polynomials.

Definition 2.3. *The roots of $x^n - 1 \in \mathbb{F}_q[x]$ are called the n-th roots of unity over $\mathbb{F}_q$.*

The structure of the n-th roots of unity over $\mathbb{F}_q$ is the same as it is for the n-th roots of unity over $\mathbb{C}$, provided that n is coprime to the characteristic of $\mathbb{F}_q$. In this case, the set of all n-th roots forms a cyclic group of order n, and in analogy with $\mathbb{C}$, we call a generator of this group a *primitive n-th root of unity*.

We are now ready to introduce the following.

Definition 2.4. *For $n \in \mathbb{N}$ not divisible by the characteristic of $\mathbb{F}_q$, let ζ_n be a primitive n-th root of unity. Then the n-th cyclotomic polynomial over $\mathbb{F}_q$ is defined by*

$$\Phi_n(x) = \prod_{1 \le k \le n,\ \gcd(k,n)=1} (x - \zeta_n^k).$$

Note that the degree of $\Phi_n(x)$ is just $\phi(n)$. The following elementary but important result can be found in Theorem 2.45 of [92].

Theorem 2.1. *For $n \in \mathbb{N}$ not divisible by the characteristic of $\mathbb{F}_q$, we have:*

$$x^n - 1 = \prod_{d|n} \Phi_d(x).$$

Clearly, $\Phi_n(x) \mid x^n - 1$ and so the subgroup of $\mathbb{F}_{q^n}^\times$ of order $\Phi_n(q)$ embeds into $\mathbb{F}_{q^n}$. Indeed, if one assumes that $\Phi_n(q) > n$, which is always the case for cryptographic parameter sizes, then this subgroup does not embed into a proper subfield of $\mathbb{F}_{q^n}$ [85]. Similarly, for each $d|n$, the subgroup of order $\Phi_d(q)$ embeds into $\mathbb{F}_{q^d}$ and no smaller field. Hence the subgroup of order $\Phi_n(q)$ may be regarded as the 'cryptographically strongest' subgroup of $\mathbb{F}_{q^n}^\times$, and so this subgroup is always used in applications.

2.2 Algebraic Tori

In the following we shall assume the reader has at least an elementary understanding of algebraic geometry, as can be found in Chapter 1 of [61], or Chapter 1 of [144], for example.

Let $G_{q,n}$ denote the subgroup of $\mathbb{F}_{q^n}^\times$ of order $\Phi_n(q)$. The starting point for torus-based cryptography is the observation due to Rubin and Silverberg [127] that $G_{q,n}$ can be identified with an algebraic torus - a perspective which as we will see during the course of the thesis allows new ideas to enter finite field arithmetic, representation, and discrete logarithm algorithms.

2.2.1 The Torus $T_n(\mathbb{F}_q)$

Definition 2.5. *Let $k = \mathbb{F}_q$ and $L = \mathbb{F}_{q^n}$. The torus T_n is the intersection of the kernels of the norm maps $N_{L/F}$, for all subfields $k \subset F \subsetneq L$:*

$$T_n(k) := \bigcap_{k \subset F \subsetneq L} \mathrm{Ker}[N_{L/F}].$$

The following lemma provides some relevant properties of T_n [127]:

Lemma 2.1. *1. $T_n(\mathbb{F}_q) \cong G_{q,n}$, and thus $\#T_n(\mathbb{F}_q) = \Phi_n(q)$;*

2. If $h \in T_n(\mathbb{F}_q)$ is an element of prime order not dividing n, then h does not lie in a proper subfield of $\mathbb{F}_{q^n}/\mathbb{F}_q$.

Note that part 2 of Lemma 2.1 essentially restates what was noted at the end of Section 2.1.2, namely, that the security of $T_n(\mathbb{F}_q)$ is essentially equivalent to the security of $\mathbb{F}_{q^n}^\times$ (see Chapter 9 for a full justification of this fact).

2.2.2 Rationality of Tori over $\mathbb{F}_q$

In order to compress elements of the variety T_n, we make use of rationality, for particular values of n. The rationality of T_n means there exists a birational map from T_n to $\phi(n)$-dimensional affine space $\mathbb{A}^{\phi(n)}$. This allows one to represent nearly all elements of $T_n(\mathbb{F}_q)$ with just $\phi(n)$ elements of $\mathbb{F}_q$, providing an effective compression factor of $n/\phi(n)$ over the embedding of $T_n(\mathbb{F}_q)$ into $\mathbb{F}_{q^n}$. Since T_n has dimension $\phi(n)$, this compression factor is optimal. T_n is known to be rational when n is either a prime power, or is a product of two prime powers, and is conjectured to be rational for all n [161]. This may seem somewhat bizarre, since it has not been verified in a single other case. Nevertheless, formally, rationality can be defined as follows.

Definition 2.6. *Let T_n be an algebraic torus over $\mathbb{F}_q$ of dimension $d = \phi(n)$, then T_n is said to be* rational *if there is a birational map $\rho : T_n \to \mathbb{A}^{\phi(n)}$ defined over $\mathbb{F}_q$.*

That is, there are Zariski open subsets $W \subset T_n$ and $U \subset \mathbb{A}^{\phi(n)}$, and rational functions $\rho_1, \ldots, \rho_{\phi(n)} \in \mathbb{F}_q(x_1, \ldots, x_n)$ and $\psi_1, \ldots, \psi_n \in \mathbb{F}_q(y_1, \ldots, y_{\phi(n)})$ such that $\rho = (\rho_1, \ldots, \rho_{\phi(n)}) : W \to U$ and $\psi = (\psi_1, \ldots, \psi_n) : U \to W$ are inverse isomorphisms. Furthermore, since W and U are open, the differences $T_n \setminus W$ and $\mathbb{A}^{\phi(n)} \setminus U$ are varieties of dimension $\leq (d-1)$, which implies that W (resp. U) is ‘almost the whole’ of T_n (resp. $\mathbb{A}^{\phi(n)}$).

2.2.3 CEILIDH

The cryptosystem CEILIDH[1] is based on the rational parametrisation of the torus T_6, as described in [127]. Strictly speaking, CEILIDH is a compression and decompression mechanism attached to the standard key exchange, encryption and signature schemes one uses for arbitrary finite fields, and indeed cyclic groups, at least as first described.

In this section we detail how one can obtain a rational parametrisation of T_6 (which also provides a rational parametrisation for T_2 *en passant*), which is taken virtually unabridged from [127].

Fix $x \in \mathbb{F}_{q^2} \setminus \mathbb{F}_q$, so $\mathbb{F}_{q^2} = \mathbb{F}_q(x)$, and let $\{\alpha_1, \alpha_2, \alpha_3\}$ be a basis for $\mathbb{F}_{q^3}$ over $\mathbb{F}_q$. Then $\{\alpha_1, \alpha_2, \alpha_3, x\alpha_1, x\alpha_2, x\alpha_3\}$ is a basis for $\mathbb{F}_{q^6}$ over $\mathbb{F}_q$. Let $\sigma \in$

[1]CEILIDH, pronounced ‘*kayley*’, is derived from the acronym for Compact, Effi cient, Improves on LUC and Improves on Diffi e-Hellman.

$\mathrm{Gal}(\mathbb{F}_{q^6}/\mathbb{F}_q)$ be the element of order two. Define $\psi_0 : \mathbb{A}^3(\mathbb{F}_q) \hookrightarrow \mathbb{F}_{q^6}$ by

$$\psi_0(u_1, u_2, u_3) = \frac{\gamma + x}{\gamma + \sigma(x)},$$

where $\gamma = u_1\alpha_1 + u_2\alpha_2 + u_3\alpha_3$. Then $N_{\mathbb{F}_{q^6}/\mathbb{F}_{q^3}}(\psi_0(\mathbf{u})) = 1$ for every $\mathbf{u} = (u_1, u_2, u_3)$. Let $U = \{\mathbf{u} \in \mathbb{A}^3 : N_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(\psi_0(\mathbf{u})) = 1\}$. By Definition 2.5, $\psi_0(\mathbf{u}) \in T_6(\mathbb{F}_q)$ if and only if $\mathbf{u} \in U$, so restricting ψ_0 to U gives a morphism $\psi_0 : U \longrightarrow T_6$. It follows from Hilbert's Theorem 90 that every element of $T_6(\mathbb{F}_q) \setminus \{1\}$ is in the image of ψ_0, and so ψ_0 defines an isomorphism

$$\psi_0 : U \xrightarrow{\sim} T_6 \setminus \{1\}.$$

The equation defining U is a quadratic hypersurface in u_1, u_2, u_3. Fix a point $\mathbf{a} = (a_1, a_2, a_3) \in U(\mathbb{F}_q)$. By adjusting the basis $\{\alpha_1, \alpha_2, \alpha_3\}$ of $\mathbb{F}_{q^3}$ if necessary, one can assume without loss of generality that the tangent plane at $\mathbf{a}$ to the surface U is just the plane $u_1 = a_1$. If $(v_1, v_2) \in \mathbb{F}_q \times \mathbb{F}_q$, then the intersection of U with the line $\mathbf{a} + t(1, v_1, v_2)$ consists of two points, namely $\mathbf{a}$ and a point of the form $\mathbf{a} + \frac{1}{f(v_1,v_2)}(1, v_1, v_2)$ where $f(v_1, v_2) \in \mathbb{F}_q[v_1, v_2]$ is an explicit polynomial independent of q. The map that takes (v_1, v_2) to the latter point is a birational isomorphism

$$g : \mathbb{A}^2 \setminus V(f) \xrightarrow{\sim} U \setminus \{\mathbf{a}\},$$

where $V(f)$ denotes the subvariety of $\mathbb{A}^2$ defined by $f(v_1, v_2) = 0$. Thus $\psi_0 \circ g$ defines an isomorphism

$$\psi : \mathbb{A}^2 \setminus V(f) \xrightarrow{\sim} T_6 \setminus \{1, \psi_0(\mathbf{a})\}.$$

For the inverse isomorphism, suppose that $\beta = \beta_1 + \beta_2 x \in T_6(\mathbb{F}_q) \setminus \{1, \psi_0(\mathbf{a})\}$ with $\beta_1, \beta_2 \in \mathbb{F}_{p^3}$. One can check that $\beta_2 \neq 0$, and if $\gamma = (1 + \beta_1)/\beta_2$, then $(\gamma + x)/\sigma(\gamma + x) = \beta$. Write $(1 + \beta_1)/\beta_2 = u_1\alpha_1 + u_2\alpha_2 + u_3\alpha_3$ with $u_i \in \mathbb{F}_q$, and define

$$\rho(\beta) = \left(\frac{u_2 - a_2}{u_1 - a_1}, \frac{u_3 - a_3}{u_1 - a_1} \right).$$

Then $\rho : T_6(\mathbb{F}_q) \setminus \{1, \psi_0(\mathbf{a})\} \xrightarrow{\sim} \mathbb{A}^2(\mathbb{F}_q) \setminus V(f)$ is the inverse isomorphism of ψ, and hence we have an efficient compression and decompression mechanism for all (bar two) elements of $T_6(\mathbb{F}_q)$.

2.2.4 XTR

In common with CEILIDH, XTR[2] is based on the cyclotomic subgroup of $\mathbb{F}_{q^6}^{\times}$ of order $\Phi_n(q)$, i.e., the torus T_6 - though this interpretation was not given (nor needed) for its original exposition and subsequent development [88, 89, 149].

Let g be a generator for $G_{q,6}$. Rather than using rational maps to affine space, in XTR elements of $\langle g \rangle$ are represented by their trace over $\mathbb{F}_{q^2}$

$$Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(g) = g + g^{q^2} + g^{q^4} \in \mathbb{F}_{q^2},$$

and hence need only two elements of $\mathbb{F}_q$ to specify. The set of traces constitute the XTR 'group'. Clearly,

$$Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(g) = Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(g^{q^2}) = Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(g^{q^4}),$$

and so given an element in the XTR group one can not distinguish between g and its conjugates g^{q^2} and g^{q^4}. Hence decompression to $\mathbb{F}_{q^6}$ is not unique, though this can easily be resolved. The analogue of the DLP is to compute n given $Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(g)$ and $Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(g^n)$. One can convert this to an ordinary DLP by mapping both back to $\mathbb{F}_{q^6}^{\times}$ by finding the correct root of

$$X^3 - Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(g)X^2 + Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(g)^q X - 1 = (X - g)(X - g^{q^2})(X - g^{q^4}),$$

and similarly for $Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(g^n)$. The real benefit of the XTR representation is the speed with which arithmetic can be performed. Let $c_n = Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(g^n)$. To compute c_n given c_1, one uses some properties of third order addition chains over

[2]XTR, pronounced '*X-T-R*', is the phonetic pronunciation of the acronym ECSTR, which stands for Effi cient Compact Subgroup Trace Representation.

$\mathbb{F}_{q^2}$ applied to the recurrence

$$c_{u+v} = c_u c_v - c_u^q c_{u-v} + c_{u-2v}.$$

As a consequence, exponentiations can be performed faster than with an optimal representation of $\mathbb{F}_{q^6}$ [148, 149]. The main drawback of XTR is that one can not perform straightforward multiplication since the set of traces is not a group (in contrast with torus-based cryptography, since T_n is a group). By keeping track of the correct conjugate however, this can be accomplished. Rather than being based on the torus T_6, XTR may in fact be viewed algebraically as a quotient of T_6 by an action of the symmetric group S_3 [127]. What makes XTR possible is that the trace map from this quotient variety to $\mathbb{F}_{q^2}$ provides an explicit rational parametrisation.

2.3 Elliptic Curves

Elliptic curves have been studied for many centuries. Indeed, there exists an extensive literature on their properties, with their study constituting a significant amount of modern research. We present only a parsimonious introduction which covers the background essential for our cryptographic purposes. A good introduction to the arithmetic of elliptic curves is [144], and excellent references for the cryptographic issues related to elliptic curves are [10, 11].

2.3.1 Background

Elliptic curves are curves of genus one, and can be described in a variety of ways. Following Weierstrass, for an arbitrary field K, they can be defined as the set of solutions in the projective plane $\mathbb{P}^2(\overline{K})$ of a homogeneous *Weierstrass equation* of the form

$$E : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3, \qquad (2.1)$$

with $a_1, a_2, a_3, a_4, a_6 \in K$.

There are many other ways to define elliptic curves using weighted projective coordinates, see for example [10, 64]. Whatever the defining equation, in order to be non-degenerate, i.e., of genus one rather than genus zero, such a curve should be non-singular in the sense that, if the equation is written in the form $F(X, Y, Z) = 0$, then the partial derivatives of the curve equation $\partial F/\partial X$, $\partial F/\partial Y$, and $\partial F/\partial Z$ should not vanish simultaneously at any point on the curve.

One can alternatively use the *affine* version of the Weierstrass equation (2.1), given by

$$E\,:\,Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6, \tag{2.2}$$

where again $a_i \in K$.

Regardless of the defining equation, for a field $\hat{K}$ such that $K \subset \hat{K} \subset \overline{K}$, one refers to as *$\hat{K}$-rational points*, those points in $\mathbb{P}^2(\hat{K})$ or $\mathbb{A}^2(\hat{K})$ that satisfy the respective curve equation, or when the field of definition of the curve is understood, these points are simply referred to as *rational points*. For any field $\hat{K}$ we denote the set of $\hat{K}$-rational points on E by $E(\hat{K})$.

In the projective case, the curve has exactly one rational point with coordinate Z equal to zero, namely $(0, 1, 0)$. This is the *point at infinity*, which we denote by $\mathcal{O}$. The K-rational points in the affine case are the solutions to E in $\mathbb{A}^2(K)$, and the point at infinity $\mathcal{O}$. To transfer between the projective and affine versions of E, one uses the following correspondence. For $Z \neq 0$, a projective point (X, Y, Z) satisfying Equation (2.1) corresponds to the affine point $(X/Z, Y/Z)$ satisfying Equation (2.2), with the inverse map from affine to projective space simply $(X, Y) \mapsto (X, Y, 1)$, or any $K^\times$-multiple thereof.

2.3.2 The Group Law

A fundamental fact in algebraic geometry is that to any smooth irreducible algebraic curve one can associate a corresponding group structure, namely, the divisor class group [61]. However for an elliptic curve, one can obtain a very simple group law without resorting to the language of divisors at all. Using the simple observation that a line intersecting two K-rational points on E will intersect E

at precisely one other K-rational point (since a line intersects a cubic at precisely three points, and the line, the curve and the two initial points are all defined over K as well), one can endow the set of rational points $E(K)$ on the curve E with a natural group structure. Much of the arithmetic theory of elliptic curves arises from the study of this group. In this section we recall the group law for points in $E(K)$, and give explicit formulae for how the group operation is computed.

Assuming for convenience that char$(K) \neq 2, 3$, one can, with a linear transformation of variables, write the defining equation for E as

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

for some $a, b \in K$. This form is also known as the *short Weierstrass form* of E.

Assuming for ease of visualisation that $K = \mathbb{R}$, suppose we wish to ‘add’ two distinct points P and Q in $E(\mathbb{R})$. We draw a straight line passing through both P and Q, which as we observed passes through a third point R, also in $E(\mathbb{R})$. We then reflect R in the x-axis, which we define to be $P + Q$ (see Figure 2.1).

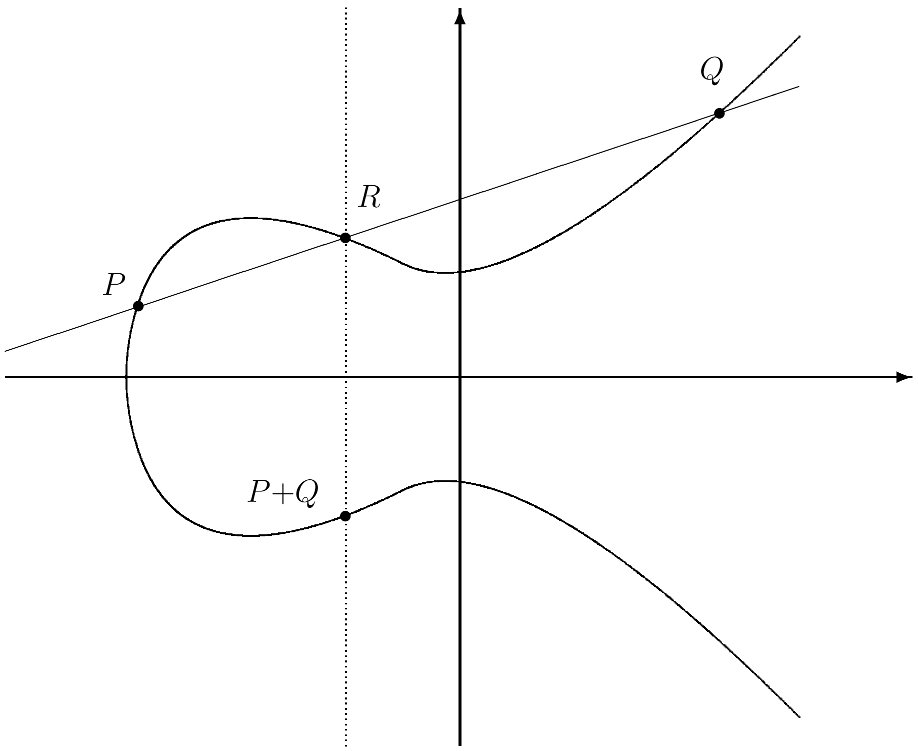
Similarly, to add a point P to itself, one performs the same operation as before but in the limit as Q tends to P, so that the straight line passing through P and Q then simply becomes the tangent line at P. Denote again by R the point on E where this line intersects the curve at a third point (counting the tangent and curve intersection as a double intersection). Then reflecting R in the x-axis gives us $2P$ (see Figure 2.2). If the tangent to the point P happens to be vertical, then the third ‘intersection’ with E is just the point at infinity, so that $P + P = \mathcal{O}$, i.e., P has order 2.

It can be shown that the process just described for adding or doubling points on E endows $E(\hat{K})$ with an abelian group structure, for any field $K \subseteq \hat{K} \subseteq \overline{K}$, with the point at infinity, $\mathcal{O}$, as the identity, or zero element [144]. Using this geometric definition, one can determine explicit algebraic formulae for the above group law, which is readily extended to all characteristics.

Lemma 2.2. *Let E denote an elliptic curve given by*

$$E : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$$

Figure 2.1: Adding two points on an elliptic curve



and let $P_1 = (x_1, y_1)$ *and* $P_2 = (x_2, y_2)$ *denote points on the curve. Then*

$$-P_1 = (x_1, -y_1 - a_1x_1 - a_3).$$

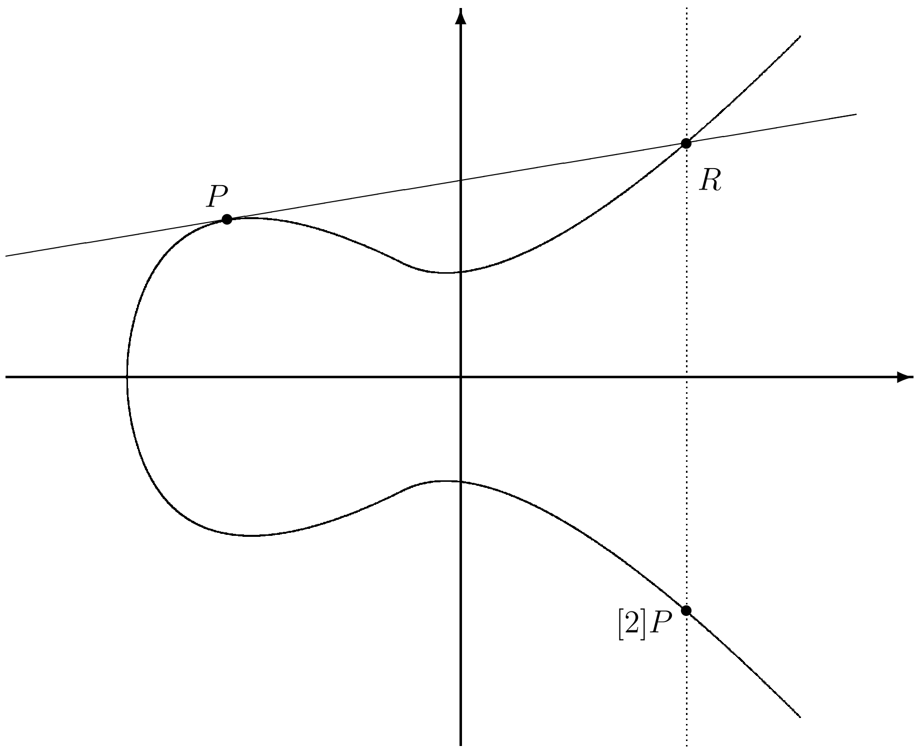
Set

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \quad , \quad \mu = \frac{y_1x_2 - y_2x_1}{x_2 - x_1}$$

when $x_1 \neq x_2$*, and set*

$$\lambda = \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} \quad , \quad \mu = \frac{-x_1^3 + a_4x_1 + 2a_6 - a_3y_1}{2y_1 + a_1x_1 + a_3}$$

Figure 2.2: Doubling a point on an elliptic curve



when $x_1 = x_2$ *and* $P_2 \neq -P_1$. *If*

$$P_3 = (x_3, y_3) = P_1 + P_2 \neq \mathcal{O}$$

then x_3 *and* y_3 *are given by the formulae*

$$\begin{aligned} x_3 &= \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \\ y_3 &= -(\lambda + a_1)x_3 - \mu - a_3. \end{aligned}$$

2.3.3 Elliptic Curves over Finite Fields

As one would expect, the theory of elliptic curves over finite fields is far simpler than it is in the characteristic zero case. Over $\mathbb{F}_q$, the number of $\mathbb{F}_q$-rational points on an elliptic curve E is clearly finite, and we denote this number by $\#E(\mathbb{F}_q)$. The quantity t defined by

$$\#E(\mathbb{F}_q) = q + 1 - t$$

is called the *trace of Frobenius* at q.

A first approximation to the order of $E(\mathbb{F}_q)$ is given by the well known theorem of Hasse, which states that $|t| \leq 2\sqrt{q}$, a proof of which can be found in [144, Theorem V.1.1]. To be of use in cryptography, it is essential to be able to construct or find elliptic curves over a given field with suitable group orders, and exploiting the Frobenius map allows for sophisticated methods to be applied [158].

The basic operation underlying elliptic curve cryptography is point multiplication. For a positive integer k, the multiplication-by-k map is a function which takes inputs k and a point $P \in E(\mathbb{F}_q)$, and outputs $kP = P + P + \cdots + P$ (k summands). For negative k one defines kP to be $(-k)(-P)$, as one would expect. The corresponding discrete logarithm problem, the ECDLP, is to compute k given P and kP. Except for some easily identifiable curves, for large k this problem is believed to be hard, and forms the basis of all elliptic curve cryptographic security.

2.4 Pairings

Pairings, or more correctly, non-degenerate bilinear maps, form the mathematical basis for all efficient identity-based cryptosystems. For use in cryptography, it is important that they can be computed efficiently. This area has developed significantly from the time Miller first wrote down how one compute pairings on elliptic curves over finite fields [102].

There are essentially two pairings that may be used for implementations in this scenario: the Weil, and Tate pairings. The latter is usually preferred since it seems to be more efficiently implementable than the former, and so we describe only the Tate pairing here. Note that recent work by Koblitz and Menezes suggests that contrary to this common assumption, for much higher security levels the reverse may be true [76].

2.4.1 The Tate Pairing

The Tate pairing on an elliptic curve is usually computed using a variant of Miller's algorithm [102]. For the special curves often used in cryptography however, it was shown independently by Barreto *et al.* [7] and Galbraith *et al.* [43] that much of the computation of the algorithm is redundant, making the use of identity-based cryptosystems far more attractive. One small issue is that the output of the Tate pairing belongs to a quotient group for which element representation is not unique. For applications one therefore uses some method to remove this ambiguity, yielding what is referred to as the *reduced Tate pairing*.

Let E be an elliptic curve over a finite field $\mathbb{F}_q$, and let $\mathcal{O}_E$ denote the identity element of the associated group of rational points on $E(\mathbb{F}_q)$, i.e., the point at infinity. For a positive integer $l \mid \#E(\mathbb{F}_q)$ coprime to q, let $\mathbb{F}_{q^k}$ be the smallest extension field of $\mathbb{F}_q$ which contains the l-th roots of unity in $\overline{\mathbb{F}}_q$. Also, let $E(\mathbb{F}_q)[l]$ denote the subgroup of $E(\mathbb{F}_q)$ of all points of order dividing l. From an efficiency perspective, k is usually chosen to be even [7]. For a fuller treatment of the following, we refer the reader to [11], and to [144] for an introduction to divisors. Then assuming that $l^2 \nmid \#E(\mathbb{F}_{q^k})$, the reduced Tate pairing of order l is the map

$$e_l : E(\mathbb{F}_q)[l] \times E(\mathbb{F}_{q^k})/lE(\mathbb{F}_{q^k}) \rightarrow \mu_l \subset \mathbb{F}_{q^k}^{\times},$$

given by $e_l(P,Q) = f_{P,l}(\mathcal{D})^{(q^k-1)/l}$, where μ_l is the set of l-th roots of unity over $\mathbb{F}_q$. Here $f_{P,l}$ is a function on E whose divisor is equivalent to $l(P) - l(\mathcal{O}_E)$, $\mathcal{D}$ is a divisor equivalent to $(Q) - (\mathcal{O}_E)$, whose support is disjoint from the support of $f_{P,l}$, and $f_{P,l}(\mathcal{D}) = \prod_i f_{P,l}(P_i)^{a_i}$, where $\mathcal{D} = \sum_i a_i P_i$. It satisfies the following properties [11]:

- For each $P \neq \mathcal{O}_E$ there exists $Q \in E(\mathbb{F}_{q^k})/lE(\mathbb{F}_{q^k})$ such that $e_l(P,Q) \neq 1$

(*non-degeneracy*);

- For any integer n, $e_l([n]P, Q) = e_l(P, [n]Q) = e_l(P, Q)^n$ for all $P \in E(\mathbb{F}_q)[l]$ and $Q \in E(\mathbb{F}_{q^k})[l]$ (*bilinearity*);
- Let $L = hl$. Then $e_l(P, Q)^{(q^k-1)/l} = e_L(P, Q)^{(q^k-1)/L}$.

When one computes $f_{P,l}(\mathcal{D})$, the value obtained belongs to the quotient group $\mathbb{F}_{q^k}^{\times}/(\mathbb{F}_{q^k}^{\times})^l$, and not $\mathbb{F}_{q^k}^{\times}$. In this quotient, for a and b in $\mathbb{F}_{q^k}^{\times}$, $a \sim b$ if and only if there exists $c \in \mathbb{F}_{q^k}^{\times}$ such that $a = bc^l$. Clearly, this is equivalent to

$$a \sim b \text{ if and only if } a^{(q^k-1)/l} = b^{(q^k-1)/l},$$

and hence one ordinarily uses this value as the canonical representative of each coset. The isomorphism between $\mathbb{F}_{q^k}^{\times}/(\mathbb{F}_{q^k}^{\times})^l$ and the elements of order l in $\mathbb{F}_{q^k}^{\times}$ given by this exponentiation makes it possible to compute $f_{P,l}(Q)$ rather than $f_{P,l}(\mathcal{D})$ [7]. It also removes the need to compute the costly denominators in Miller's algorithm. In Chapter 4 we give an overview of some of the techniques employed to evaluate pairings efficiently.

Part I

Arithmetics

Chapter 3

A Comparison of CEILIDH and XTR

In this chapter we develop efficient arithmetic for the torus-based cryptosystem CEILIDH, and compare the resulting performance with XTR.

This chapter represents joint work with Dan Page and Martijn Stam, and appeared in [56]. We thank Fréderik Vercauteren for suggesting this research, and for many fruitful discussions.

3.1 Introduction

Underpinning both CEILIDH [127] and XTR [88] is the mathematics of the two dimensional algebraic torus T_6. However, while they both attain the same discrete logarithm security and each achieve a compression factor of three for all data transmissions, the arithmetic performed in each is fundamentally different. In its inception, the designers of CEILIDH were reluctant to claim it offers any particular advantages over XTR other than its exact compression and decompression technique. From both an algorithmic and arithmetic perspective, in this chapter we develop an efficient version of CEILIDH and show that while it seems bound to be inherently slower than XTR, the difference in performance is much smaller than one might infer from the original description. Also, thanks to CEILIDH's simple group law, it provides a slightly greater flexibility for applications, and may thus be considered a worthwhile alternative to XTR.

In both the following exposition and our implementation, we take as our base field $\mathbb{F}_p$, for two reasons. Firstly, as originally described, XTR is specified over prime fields [88] (although extensions have been suggested [93]). Secondly, this choice allows keys to be generated exactly as in XTR, which permits a fair comparison. The restriction to prime fields is therefore artificial, and naturally any finite field would suffice.

3.2 Efficient Representations for $T_6(\mathbb{F}_p)$

In this section we develop suitable field representations and algorithms for the efficient implementation of the CEILIDH cryptosystem. We base our implementation on Example 11 of [127], since the rational maps specified are (clearly) convenient to write down. Our main result is a multiplication-efficient representation of T_6, but we also make some basic observations that result in a considerable improvement over the original arithmetic.

Depending on the particular protocol one wishes to implement, and even on which part of the protocol, some representations of the underlying field $\mathbb{F}_{p^6}$ may perform better than others. For example, in the first stage of a Diffie-Hellman key agreement, both parties exponentiate a fixed public base g, so here one can precompute some powers of g, and one should use a field representation that permits fast multiplication. In the second stage, both parties exponentiate a random element of $\mathbb{F}_{p^6}^{\times}$, and so here one should use a representation which permits fast squaring. As we show, optimising each consideration leads to different field representations, whilst switching between them is a simple matter.

For the two operations of exponentiating a fixed and a random base, two field representations suffice. The first is a degree six extension of $\mathbb{F}_p$ and allows us to use many implementation tricks [148], which include very fast squaring in T_6. We refer to this representation as F_1, the details of which we present in Section 3.2.1.

The second representation F_2, presented in Section 3.2.2, fulfills two functions: built as a quadratic extension of a cubic extension of $\mathbb{F}_p$, it firstly permits the efficient use of the birational maps ψ and ρ which are essential to CEILIDH, given our cheap inversion method based on the Frobenius automorphism; primarily however, it provides the basis for arithmetic in F_3.

What we refer to as F_3, presented in Section 3.2.3, is a semi-compressed fractional form of the torus T_6, which is in fact the projective version of the torus T_2. Its utility is that one can perform the group operation, but with much better multiplication efficiency. Together, F_1, F_2, F_3, $\mathbb{A}^2$ and the isomorphisms between them constitute our implementation of CEILIDH, which may be depicted as follows:

$$\begin{tikzcd} F_1 \arrow[r, shift left, "\sigma"] & F_2 \arrow[l, shift left, "\sigma^{-1}"] \arrow[r, shift left, "\tau"] & F_3 \arrow[l, shift left, "\tau^{-1}"] \arrow[r, shift left, "\rho"] & \mathbb{A}^2(\mathbb{F}_p). \arrow[l, shift left, "\psi"] \end{tikzcd}$$

For F_1 we give a brief description of the arithmetic, and for F_2 and F_3 we give full details of all operations. In Lemma 3.2 we provide a simple cost analysis where M,A, and I represent the cost of an $\mathbb{F}_p$ multiplication, addition, and inversion respectively. For our operation counts, we assume that a subtraction in $\mathbb{F}_p$ costs the same as an addition, and also that squaring costs the same as a multiplication, since the former operation is seldom used.

For $(n, p) = 1$, let ζ_n denote a primitive n-th root of unity over $\mathbb{F}_p$, and as in XTR let $p \equiv 2 \bmod 9$ throughout ($p \equiv 5 \bmod 9$ is equally valid).

3.2.1 The Representation F_1

A full derivation of the results of this section can be found in [148]. Let $z = \zeta_9$, so that $\mathbb{F}_{p^6} = \mathbb{F}_p(z)$, and let our basis for $\mathbb{F}_{p^6}$ be $\{z, z^2, z^3, z^4, z^5, z^6\}$. Using a Karatsuba-type method for multiplication and squaring, these can be performed in $18M + 53A$ and $12M + 42A$ respectively. However, working entirely within T_6, improvements can be made. For example, if $g \in T_6$, inversion is just the cube of the Frobenius automorphism, since $\Phi_6(p) = (p^2 - p + 1) \mid (p^3 + 1)$, and so $g^{-1} = g^{p^3}$. Also the condition $g^{\Phi_6(p)} = 1$ gives a set of six equations on the six coefficients of g, which remarkably enables squaring to be performed with a cost of just $6M + 21A$, see [148].

3.2.2 The Representation F_2

Let $x = \zeta_3$ and $y = \zeta_9 + \zeta_9^{-1}$. Then $\mathbb{F}_{p^3} = \mathbb{F}_p(y)$, and $\mathbb{F}_{p^6} = \mathbb{F}_{p^3}(x)$. The bases we use are $\{1, y, y^2 - 2\}$ for $\mathbb{F}_{p^3}$, and $\{1, x\}$ for the degree two extension. Note that the minimal polynomials for y and x are $y^3 - 3y - 1 = 0$, and $x^2 + x + 1$ respectively. We now describe the basic arithmetic in each of these extensions.

$\mathbb{F}_{\mathbf{p^3}}$ **Frobenius** :
For our basis, since $p \equiv 2 \bmod 9$, the Frobenius map gives $y^p = y^2 - 2$, and $(y^2 - 2)^p = -y - (y^2 - 2)$. Hence for $a = a_0 + a_1 y + a_2(y^2 - 2)$, $a^p = a_0 - a_2 y + (a_1 - a_2)(y^2 - 2)$.

$\mathbb{F}_{\mathbf{p^3}}$ **Multiplication** :
Let $a = a_0 + a_1 y + a_2(y^2 - 2)$, $b = b_0 + b_1 y + b_2(y^2 - 2)$. Then $ab = (a_0 b_0 + 2a_1 b_1 + 2a_2 b_2 - a_1 b_2 - a_2 b_1) + (a_0 b_1 + a_1 b_0 + a_1 b_2 + a_2 b_1 - a_2 b_2)y + (a_0 b_2 + a_2 b_0 + a_1 b_1 - a_2 b_2)(y^2 - 2)$. Precompute $t_{00} = a_0 b_0$, $t_{11} = a_1 b_1$, $t_{22} = a_2 b_2$, and $t_{01} = (a_0 + a_1)(b_0 + b_1)$, $t_{12} = (a_1 - a_2)(b_2 - b_1)$, $t_{20} = (a_2 - a_0)(b_0 - b_2)$. Then $ab = (t_{00} + t_{11} + t_{22} - t_{12}) + (t_{01} + t_{12} - t_{00})y + (t_{20} + t_{00} + t_{11})(y^2 - 2)$.

$\mathbb{F}_{\mathbf{p^3}}$ **Inversion** :
Usually, to invert an element in an extension field one must either use a GCD algorithm on the polynomial representation, or one can simply exponentiate to a power one less than the group order. However, since the extension degree is small, we can perform inversion directly, reducing it to just one inversion in $\mathbb{F}_p$ (along with a few other operations): one uses the multiplication formula and sets the result to the identity, i.e., one solves

$$\begin{pmatrix} a_0 & 2a_1 - a_2 & 2a_2 - a_1 \\ a_1 & a_0 + a_2 & a_1 - a_2 \\ a_2 & a_1 & a_0 - a_2 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

for b. This gives

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = \tfrac{1}{\Delta} \begin{pmatrix} -a_0^2 + a_1^2 + a_2^2 - a_1 a_2 \\ a_2^2 + a_0 a_1 - 2a_1 a_2 \\ -a_1^2 + a_2^2 + a_0 a_2 \end{pmatrix},$$

where $\Delta = -a_0^3 + a_1^3 + a_2^3 + 3a_0a_2^2 + 3a_0a_1^2 + 3a_1a_2^2 - 6a_1^2a_2 - 3a_0a_1a_2$. Computing $t_{00} = a_0^2, t_{11} = a_1^2, t_{22} = a_2^2, t_{01} = a_0a_1, t_{12} = a_1a_2, t_{20} = a_2a_0, t_{012} = a_0 + a_1 + a_2$ and $t = t_{12}(a_0 + a_1)$, then $\Delta = t_{012}^3 - t_{00}(3t_{012} - a_0) - 9t$. To finish, we perform one $\mathbb{F}_p$ inversion and obtain a^{-1} equals

$$\Delta^{-1}((t_{11} + t_{22} - t_{00} - t_{12}) + (t_{01} - 2t_{12} + t_{22})y + (t_{20} + t_{22} - t_{11})(y^2 - 2)).$$

$\mathbb{F}_{\mathbf{p^6}}$ **Frobenius** :
Let $c = c_0 + c_1x$. Then $c^p = (c_0 + c_1x)^p = (c_0^p - c_1^p) - c_1^p x$.

$\mathbb{F}_{\mathbf{p^6}}$ **Multiplication** :
For $c = c_0 + c_1x$, $d = d_0 + d_1x$, we have $cd = (c_0d_0 - c_1d_1) + (c_0d_1 + c_1d_0 - c_1d_1)x$. If we compute $t_{00} = c_0d_0$, $t_{11} = c_1d_1$, and $t_{01} = (c_0 + c_1)(d_0 + d_1)$, then $cd = (t_{00} - t_{11}) + (t_{01} - t_{00} - 2t_{11})x$.

$\mathbb{F}_{\mathbf{p^6}}$ **Squaring** :
$c^2 = (c_0 + c_1x)^2 = (c_0^2 - c_1^2) + c_1(2c_0 - c_1)x$. We compute $t_{01} = (c_0 + c_1)(c_0 - c_1)$, giving $c^2 = t_{01} + c_1(2c_0 - c_1)x$.

$\mathbb{F}_{\mathbf{p^6}}$ **Inversion** :
Performing again a direct inversion as in $\mathbb{F}_{p^3}$, we find

$$d_0 + d_1x = (c_0 + c_1x)^{-1} = \frac{1}{c_0^2 - c_0c_1 + c_1^2} \begin{pmatrix} c_1 - c_0 \\ -c_1 \end{pmatrix},$$

so that we still only require one $\mathbb{F}_p$ inversion. Precomputing $t_0 = (c_1 - c_0)$, $t_{01} = c_0c_1$, and $\Delta = t_0^2 + t_{01}$, the coefficients of the inverse d are given by $d_0 = \Delta^{-1}t_0$, $d_1 = -\Delta^{-1}c_1$.

If we are working in $T_6(\mathbb{F}_p)$, then as in F_1 inversions are essentially free thanks to the cheap Frobenius automorphism.

$\sigma : \mathbf{F_1} \rightarrow \mathbf{F_2}$:

In addition to the individual arithmetic of F_1 and F_2 we need to specify an efficiently computable isomorphism between them. Writing x and y in terms of z we find that $x = z^3$, and $y = z - z^2 - z^5$, and so $\sigma^{-1} : F_2 \rightarrow F_1$ can be evaluated with just a few additions:

$$\sigma^{-1} = \begin{pmatrix} 0 & 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} .$$

Since σ^{-1} has determinant three, a naive evaluation of σ necessitates four divisions by three. It is not possible to eliminate all of these since for our F_1, writing $\mathbb{F}_{p^6}$ as a quadratic extension of a cubic extension, all bases have determinant divisible by three. We can reduce this to just one division by three however, (or a multiplication by its precomputed inverse) by writing

$$\sigma = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ \frac{2}{3} & \frac{1}{3} & 0 & -\frac{1}{3} & \frac{1}{3} & 0 \end{pmatrix} .$$

3.2.3 The Representation F_3

In our notation, the group operation in CEILIDH as originally described is performed in F_2 [127], and the inverse birational maps $\psi : \mathbb{A}^2(\mathbb{F}_p) \setminus V(f) \xrightarrow{\sim} T_6(\mathbb{F}_p) \setminus \{1, x^2\}$, and $\rho : T_6(\mathbb{F}_p) \setminus \{1, x^2\} \xrightarrow{\sim} \mathbb{A}^2(\mathbb{F}_p) \setminus V(f)$, are given by

$$\psi(\alpha_1, \alpha_2) = \frac{1 + \alpha_1 y + \alpha_2(y^2 - 2) + (1 - \alpha_1^2 - \alpha_2^2 + \alpha_1\alpha_2)x}{1 + \alpha_1 y + \alpha_2(y^2 - 2) + (1 - \alpha_1^2 - \alpha_2^2 + \alpha_1\alpha_2)x^2}, \tag{3.1}$$

where $V(f)$ is the set of zeros of $f(\alpha_1, \alpha_2) = 1 - \alpha_1^2 - \alpha_2^2 + \alpha_1\alpha_2 = 0$ in $\mathbb{A}^2(\mathbb{F}_p)$; and for $\beta = \beta_1 + \beta_2 x$, with $\beta_1, \beta_2 \in \mathbb{F}_{p^3}$, let $(1 + \beta_1)/\beta_2 = u_1 + u_2 y + u_3(y^2 - 2)$. Then $\rho(\beta) = (u_2/u_1, u_3/u_1)$.

Since the torus T_6 is two-dimensional, given compressed points $P_1 = (\alpha_1, \alpha_2)$ and $P_2 = (\beta_1, \beta_2)$, it would be aesthetically appealing to compute their composition directly without having to map the affine representation back to $\mathbb{F}_{p^6}$, i.e., to find the $(\gamma_1, \gamma_2) \in \mathbb{A}_2$ such that $(\alpha_1, \alpha_2) \circ (\beta_1, \beta_2) = (\gamma_1, \gamma_2)$, where $\circ$ refers to an operation equivalent to multiplication in $\mathbb{F}_{p^6}$. The drawback with this approach is that the group law on T_6 as we have presented it is defined only in terms of the arithmetic of $\mathbb{F}_{p^6}$, and so the decompression of P_1 and P_2 before multiplication seems essential.

If one insists on representing intermediate results in their compressed form in an exponentiation for example, we found that this costs $24M + 43A + I$ for a multiplication, and $21M + 38A + I$ for a squaring. Note that these are even more costly than both a general elliptic curve add, or double for example. The reason these operations are so expensive is that the rational representation of T_6 does not lend itself favourably to performing the group law operation: this statement does not hold for the XTR representation, which is why the corresponding arithmetic is so much faster (cf. Figure 3.1).

The arithmetic developed in the previous section shows that if one alternatively performs a one-time decompression before performing an exponentiation, and a re-compression at the end of an exponentiation, then the cost of a general multiplication and squaring is only $18M + 54A$ and $12M + 33A$ respectively in F_2, and $18M + 53A$ and $6M + 21A$ respectively in F_1. Clearly decompressing to

$\mathbb{F}_{p^6}$ seems to be the better method.

This is not the whole story though. Since $T_6(\mathbb{F}_p)$ is by Definition 2.5 the intersection of the kernels of the two norm maps $N_{\mathbb{F}_{p^6}/\mathbb{F}_{p^3}}$ and $N_{\mathbb{F}_{p^6}/\mathbb{F}_{p^2}}$, if we have a good representation for either of these, we can save some work. The following lemma emphasises the relevance to CEILIDH of the parametrisation of elements in the kernel of the former norm map, which is implicit in the construction of ψ given in Section 2.2.3.

Lemma 3.1. *There is an isomorphism*

$$\tau : \mathrm{Ker}[N_{\mathbb{F}_{p^6}/\mathbb{F}_{p^3}}] \xrightarrow{\sim} \left\{ \frac{b+x}{b+x^2}, b \in \mathbb{F}_{p^3} \right\} \cup \{\mathcal{O}\},$$

with $\mathcal{O}$ is the point at infinity, and where for $a = a_0 + a_1 x \in \mathrm{Ker}[N_{\mathbb{F}_{p^6}/\mathbb{F}_{p^3}}] \setminus \{1\}$,

$$\tau(a) = \left(\frac{b+x}{b+x^2} \right),$$

with $b = (1+a_0)/a_1$ if $a_1 \neq 0$ and $b = a_0/(a_0 - 1)$ otherwise, and

$$\tau^{-1}\left(\frac{b+x}{b+x^2} \right) = \frac{b^2-1}{b^2-b+1} + \frac{2b-1}{b^2-b+1}x.$$

Proof. For $a \in \mathbb{F}_{p^6}$, $N_{\mathbb{F}_{p^6}/\mathbb{F}_{p^3}}(a) = a^{1+p^3}$, and so the kernel of this map has at most $1+p^3$ solutions. For each $(b+x)/(b+x^2), b \in \mathbb{F}_{p^3}$, the stated norm is one, and counting also the identity, which maps to the point at infinity, we have all $1+p^3$ solutions. Solving $a_0 + a_1 x = (b+x)/(b+x^2)$ for b gives the second part, while for the third, we solve the same equation for a_0, a_1, where we have used the condition that $a_0^2 + a_1^2 - a_0 a_1 = 1 \Leftrightarrow a_0 + a_1 x \in \mathrm{Ker}[N_{\mathbb{F}_{p^6}/\mathbb{F}_{p^3}}]$. □

With this we can introduce the following:

Definition 3.1. *F_3 is the set of elements*

$$\left\{ \frac{a_0 + a_1 x}{a_0 + a_1 x^2}, a_i \in \mathbb{F}_{p^3} \right\}.$$

When the coefficient a_1 of this representation equals 1, we say the element is reduced.

In order to obtain the reduced form of an element in F_3, one just computes a_0/a_1. Note that if we do not need the reduced form of an element in F_3, then evaluating τ simplifies to $(1 + a_0 + a_1 x)/(1 + a_0 + a_1 x^2)$, and no inversion is necessary. Mapping an unreduced element back to F_2, we obtain

$$\tau^{-1}\left(\frac{a_0 + a_1 x}{a_0 + a_1 x^2}\right) = \frac{a_0^2 - a_1^2}{a_0^2 - a_0 a_1 + a_1^2} + \frac{2a_0 a_1 - a_1^2}{a_0^2 - a_0 a_1 + a_1^2} x.$$

As we pointed out in Section 3.2, this construction is no more than the rational parametrisation of $T_2(\mathbb{F}_{p^3})$. As a result of the symmetry of x and x^2 in the irreducible polynomial defining the quadratic extension of F_2, for all arithmetic operations between elements of F_3, the coefficients of the numerator and those of the denominator correspond exactly. Hence we need only work with the numerator, which will in general have the form $a = a_1 + a_2 x$. For expositional purposes, in the following we still write elements of T_6 as fractions but for our implementation this is of course unnecessary.

Using this fractional form alone does not seem to offer any advantage over the F_2 representation. However, considering the exponentiation of a reduced element $(g + x)/(g + x^2)$ of T_6, we already save one $\mathbb{F}_{p^3}$ multiplication for every multiplication by this element, since

$$\left(\frac{g + x}{g + x^2}\right) \times \left(\frac{a_0 + a_1 x}{a_0 + a_1 x^2}\right) = \left(\frac{(ga_0 - a_1) + (ga_1 + a_0 - a_1)x}{(ga_0 - a_1) + (ga_1 + a_0 - a_1)x^2}\right), \qquad (3.2)$$

so we only need to compute ga_0 and ga_1, and a few additions, when our multiplier is in this form.

Furthermore, one can exploit the fractional form of elements of F_3 for squaring and inversion. Furthermore, if additions are sufficiently cheap compared to multiplications, one can actually reduce the cost of a basic $\mathbb{F}_{p^3}$ multiplication to the theoretical minimum measured in the number of $\mathbb{F}_p$ multiplications, using a Toom-Cook-style interpolation [72]. In this method one must divide by by a small constant; in F_3 one can simply ignore this. This possibility was not implemented in this chapter however.

The reason this all works is that the rational representation of elements of T_2 can be embedded efficiently as a fraction in the field extension. Noting that we need only work with the numerator, the group law can be performed directly on this compressed element.

The reduced form of an element of F_3 may be viewed as the affine representation of T_2, with the non-compressible identity element being the point at infinity. The non-reduced form of an element of F_3 corresponds to the projective representation, with identity λ for any $\lambda \in \mathbb{F}_{p^3}^\times$. This point was essentially made in [127], but without reference to its applicability to CEILIDH as well. There however, it was suggested the group law be performed entirely in $\mathbb{F}_{p^3}$, which would require an $\mathbb{F}_{p^3}^\times$ inversion for every multiplication: this would be very inefficient.

One may also represent $T_2(\mathbb{F}_p)$ as the isomorphic quotient group $\mathbb{F}_{p^2}^\times/\mathbb{F}_p^\times$. If $\mathbb{F}_{p^2} = \mathbb{F}_p(x)$, where $x = \zeta_3$, then elements are represented as $a_0 + a_1 x$, and $a_0 + a_1 x \equiv b_0 + b_1 x$ iff $a_0/a_1 = b_0/b_1$. As a consequence, elements for which $a_1 \neq 0$ can be represented by a_0/a_1, and all the arithmetic above follows *mutatis mutandis*. Indeed, this quotient group arises in the (non-reduced) Tate pairing, and this observation can be usefully exploited, see Chapter 4.

$\mathbf{F_3}$ Frobenius :
Let $a \in F_3$. Then

$$\left(\frac{a_0 + a_1 x}{a_0 + a_1 x^2}\right)^p = \left(\frac{a_0^p + a_1^p x^2}{a_0^p + a_1^p x}\right) = \left(\frac{(a_0^p - a_1^p) - a_1^p x}{(a_0^p - a_1^p) - a_1^p x^2}\right) = \left(\frac{(a_1^p - a_0^p) + a_1^p x}{(a_1^p - a_0^p) + a_1^p x^2}\right).$$

$\mathbf{F_3}$ Multiplication :
Multiplication by a reduced element is performed as in (3.2), or if by a non-reduced element, exactly as in F_2.

$\mathbf{F_3}$ Squaring :
This is performed as in F_2.

$\mathbf{F_3}$ Inversion :

This is straightforward, since elements are represented as fractions.

$$\left(\frac{a_0 + a_1 x}{a_0 + a_1 x^2}\right)^{-1} = \left(\frac{a_0 + a_1 x^2}{a_0 + a_1 x}\right) = \left(\frac{(a_1 - a_0) + a_1 x}{(a_1 - a_0) + a_1 x^2}\right).$$

Also, since we use the intermediate representation F_3 between $\mathbb{A}^2(\mathbb{F}_p)$ and F_2, we must adjust the map $\rho : F_3 \setminus \{1, x^2\} \xrightarrow{\sim} \mathbb{A}^2(\mathbb{F}_p) \setminus V(f)$. Let $\beta = (\beta_1 + \beta_2 x)/(\beta_1 + \beta_2 x^2) \in F_3$, with $\beta_1/\beta_2 = u_1 + u_2 y + u_3(y^2 - 2)$: then $\rho(\beta) = (u_2/u_1, u_3/u_1)$. We summarise the results regarding arithmetic in F_1, F_2 and F_3 in the following:

Lemma 3.2. *The cost of arithmetical operations in F_1, F_2 and F_3 are:*

Operation	F_1	F_2	F_3
Multiply	$18M + 53A$	$18M + 54A$	$18M + 54A$
Square	$6M + 21A$	$12M + 33A$	$12M + 33A$
Inverse	$2A$	$6A$	$3A$
Frobenius	$1A$	$10A$	$10A$
Reduce	*n/a*	*n/a*	$19M + 35A + I$
Mixed Mul.	*n/a*	*n/a*	$12M + 33A$

Map	Cost
$F_1 \rightarrow F_2$	$1M + 11A$
$F_2 \rightarrow F_3$	$1A$
$F_3 \rightarrow \mathbb{A}^2$	$14M + 19A + I$
$\mathbb{A}^2 \rightarrow F_3$	$2M + 3A$
$F_3 \rightarrow F_2$	$25M + 41A + I$
$F_2 \rightarrow F_1$	$8A$

Here the operation *Reduce* refers to obtaining the reduced form of an element of F_3, and a *Mixed Mul.* refers to multiplying a non-reduced element by a reduced one. The cost of the map $\rho : F_3 \rightarrow \mathbb{A}_2$ assumes the element being compressed is in non-reduced form, as this is the case after an exponentiation in both F_1, or F_3. Also, for the map $\tau^{-1} : F_3 \rightarrow F_2$ we assume that the x-coefficient is in $\mathbb{F}_p$ only as in (3.1), and not $\mathbb{F}_{p^3}$, as in practice one would only perform this operation when decompressing from $\mathbb{A}_2$ to F_1, and not from a non-reduced element.

3.3 Exponentiation

For F_1, F_2 and F_3 one can use the Frobenius map to obtain fast exponentiation. In a subgroup of order l where $l \mid (p^2 - p + 1)$, we write an exponent m as $m \equiv m_1 + m_2 p \mod l$, where m_1, m_2 are approximately half the bitlength of m [148]. One can find m_1 and m_2 very quickly having performed a one-time Gaussian two dimensional lattice basis reduction, and using this basis to find the closest vector to $(m, 0)^T$. To compute a^m for a random a, we perform a double exponentiation $a^{m_1}(a^p)^{m_2}$ using the Joint Sparse Form (JSF) of the integers m_1, m_2 [147], which on average halves the number of pairs of non-zero bits in their paired binary expansion, and Shamir's trick, originally due to Straus [150]. The use of the JSF is possible since we have virtually free inversion.

When the base of the exponentiation is fixed we also use the JSF and Shamir's trick but perform some precomputation as well. Fixed elements are important since we can spend some time and space to precompute values. This allows one to accelerate an operation if the values are reused often enough to make the cost of doing so acceptable. We store $(a^i(a^p)^j)^{2^k}$ for $i, j \in \{0, \pm 1\}$, and k from 1 to half the bit-length of l, where l is the size of the subgroup we work with. For a 1024-bit field, and with l approximately of length 160 bits, for the price of storing $4 \times 80 = 320$ field elements, we eliminate all squarings from the exponentiation routine. In F_3 the storage of 320 reduced elements requires about 22.5Kb, whereas for F_1 and F_2 this amount is doubled to 45Kb, since elements can not be reduced. In XTR only one element is precomputed (it is unclear how to exploit more precomputation). This provides nearly as good a speed up as for CEILIDH, but for XTR the cost of the precomputation is much cheaper in both time and space. For cryptosystems where the group law is just ordinary multiplication in a finite field though, one can always exchange space for time, so CEILIDH has a slight advantage here over XTR, if these resources are available. We concede that more efficient precomputation methods can be applied given our chosen level of storage [94], but are confident our method provides an accurate reflection of the possible gains resulting from this approach.

For double exponentiation we assume that both bases are random, so that no precomputation can be employed. Using the JSF for each exponent, we combined the squarings for both exponentiations while performing the multiplications separately. It is possible to make this slightly more efficient [122], and to use some precomputation, but again we are satisfied that our results are indicative of the general performance of the algorithms.

3.4 Implementation Results

To demonstrate the different performance characteristics of the three representations of CEILIDH, we constructed an implementation of the entire system, based on the algorithms described in the previous section. We also implemented the fastest algorithms for the equivalent XTR protocols [149], so that our comparison was made between the best possible implementations of both systems.

We based this implementation on a special purpose library for arithmetic in $\mathbb{F}_p$ that represents and manipulates field elements using Montgomery reduction [105]. Montgomery arithmetic facilitates fast field operations given some modulus specific precomputation [18]. This is ideal for our purposes since after key-generation the field $\mathbb{F}_p$ remains the same for all subsequent operations.

We used a GCC 3.3 compiler suite to build our implementation and ran timing experiments on a Linux based PC incorporating a 2.80 GHz Intel Pentium 4 processor. The entire system was constructed in C++ except for small assembly language fragments to accelerate operations in $\mathbb{F}_p$. We accept that further performance improvements could be made through aggressive profiling and optimisation but are confident our results are representative of the underlying algorithms and allow a comparison between them.

For our experiments we randomly chose 500 key pairs (p, l) with the field characterisitc p of length 176 bits and subgroup size l 160 bits. These parame-

Figure 3.1: Timing results for CEILIDH and XTR

Operation	Time			
	F_1	F_2	F_3	XTR
$x_R \cdot y_R$	37.8 μs	41.2 μs	41.8 μs	8.7 μs
x_R^2	16.8 μs	26.9 μs	27.7 μs	4.9 μs
Reduce	n/a	n/a	170.6 μs	n/a
Mixed Mul.	n/a	n/a	28.4 μs	n/a
x_R^n	2.99 ms	3.94 ms	3.94 ms	2.57 ms
$x_R^n \cdot y_R^m$	4.71 ms	5.75 ms	5.79 ms	2.98 ms
x_F^n	1.56 ms	1.74 ms	1.21 ms	1.49 ms
Precomp.	5.89 ms	9.33 ms	63.65 ms	1.43 ms

Mapping	Time
$F_1 \rightarrow F_2$	7.1 μs
$F_2 \rightarrow F_3$	1.9 μs
$F_3 \rightarrow \mathbb{A}^2$	161.7 μs
$F_1 \leftarrow F_2$	2.4 μs
$F_2 \leftarrow F_3$	222.8 μs
$F_3 \leftarrow \mathbb{A}^2$	5.1 μs

ters heuristically provide the equivalent of 1024 bit RSA security. With the same key pairs for both CEILIDH and XTR we performed 500 instances of each operation listed in Figure 3.1. For exponentiations, exponents were chosen randomly modulo l in all cases.

The upper table shows timings for operations pertinent to use in real cryptosystems. We use x_R to denote a random element in a given representation and x_F to represent a fixed element. Thus, x_R^n represents a single exponentiation with a random base and exponent while $x_R^n.y_R^m$ represents a double exponentiation without precomputation.

These timings are useful since they form the basis of all public key cryptographic protocols, and so we have a good idea of the comparative performance of the different representations of $T_6(\mathbb{F}_p)$, and XTR. One can see that the results are in general agreement with our arithmetic cost analysis. Indeed F_1 provides the most efficient representation when exponentiating a random element, and with precomputation, F_3 offers a slight improvement over XTR, allowing for the cost of $\rho : F_3 \rightarrow \mathbb{A}^2$ as well.

The lower table in Figure 3.1 demonstrates the cost of applying mapping operations on elements to transform them between our different representations. These results represent the time taken to map a random element in the source representation and transform it into the corresponding element in the target representation. One notes from these that it is unfortunately not advantageous to 'mix' representations, so that in a single exponentiation squaring is performed in F_1, and multiplication in F_3. This would be analogous to the use of mixed coordinate systems in elliptic curve cryptography [23].

Note that it is possible to negate the necessity of F_1 altogether by basing all arithmetic in a field of characteristic two or three, since in these fields squarings and cubings respectively are very low cost, and we need only then perform multiplication, which is very efficient in F_3. With suitable keys, this would simplify implementations of CEILIDH and far more efficient precomputation strategies could be brought to bare. Indeed, this is precisely what we do for characteristic three pairing-based cryptography in the next chapter.

Chapter 4

On Small Characteristic Algebraic Tori in Pairing-Based Cryptography

In this chapter we transfer the techniques developed in Chapter 3, to pairing-based cryptography, exploiting the simple observation that pairing values are group elements of a torus. Note that throughout the chapter we focus almost entirely on characteristic three fields.

This chapter represents joint work with Dan Page and Martijn Stam, and appeared in [57]. The authors would like to thank Paulo Barreto, Steven Galbraith, Keith Harrison, Karl Rubin, Mike Scott, Alice Silverberg, Nigel Smart and Fréderik Vercauteren for many helpful comments and fruitful discussions.

4.1 Introduction

The use of pairings in cryptography is now a well-studied area, with resulting applications to identity-based encryption, key-agreement and signature schemes [14, 129], tripartite Diffie-Hellman key-agreement [66], and short signatures [15], to name just a few amongst numerous others (see e.g. [29] for a recent survey).

To support these applications much research activity has focused on developing efficient and easily implementable algorithms for their deployment [7,30,43].

Currently[1] the fastest algorithm for pairing computation on elliptic curves is that of Duursma and Lee [30], which applies to the class of supersingular elliptic curves in characteristic three with so-called embedding degree six.

One is therefore free to use the trace-based methods found in LUC [146] and XTR [88] for post-pairing arithmetic [160], resulting in the compression of pairing outputs by a factor of two and three respectively. Scott and Barreto [137] also describe the use of traces for the computation of the pairing itself, however closer inspection of their work shows that their claim is misleading. Indeed, their method is essentially a polynomial basis transformation and hence does not offer any advantages during the computation of the pairing. Moreover, for characteristic three, we demonstrate that contrary to the claims of Scott and Barreto [137], the performance of their approach is inferior to a straightforward implementation. Thus besides pairing compression, the method they advocate does not seem to offer any benefits.

The contribution of this chapter is to achieve both efficient pairing arithmetic, and also pairing compression. Our methods are based on the simple observation that the quotient group to which the natural output of the Tate pairing belongs, may be viewed as a special representation of an algebraic torus.

Using this observation and the efficient point multiplication method developed for tori in Chapter 3, we are able to perform arithmetic with pairing values that is on average 30% faster than previous methods. This is useful, for example, in pairing-based protocols where one typically multiplies a point by an ephemeral random value. By bilinearity, this multiplication may be performed either on the curve before the pairing evaluation, or as an exponentiation in the extension field afterwards. Given that a pairing evaluation is usually several times more costly than either a point multiplication on the curve or an exponentiation in the field, if a pairing value ever needs to be reused, it is beneficial to compute it once and for

[1]After the initial submission of a paper describing the work in this chapter and the dissemination of a preprint of it, generalizations of the Duursma-Lee algorithm were found [6, 80]. In particular the η-pairing [6] can be faster than the Duursma-Lee algorithm optimized in this chapter. However, both our methods of exploiting the sparsity of the multiplicands to be discussed in Section 4.6, and the described techniques to deal with the fi nal exponentiations and manipulate the resulting pairing value, apply to these more recent results.

all and to perform each ephemeral exponentiation in the extension field.

Examples where this occurs include the the Boneh-Franklin identity-based encryption scheme [14], the identity-based signature scheme of Hess [62], and the certificate-based encryption scheme of Gentry [50].

The aforementioned compression methods can also be used during any interactive pairing-based protocol where pairing values are transmitted between parties. Such schemes include the selective-ID identity-based encryption scheme of Boneh and Boyen [12], the interactive proof of knowledge in the short group signature scheme of Boneh *et al.* [13], and various others [51, 136].

One may regard our methods as a characteristic three version of the results of Chapter 3 tailored for pairings. However they may also be used for pairings on any abelian variety possessing an even embedding degree, which for efficiency reasons is the case for all contemporary pairing algorithms. As such they may also be applied to supersingular binary elliptic curves, although we do not pursue this application here.

The remainder of the chapter is organised as follows. In the next section we briefly describe the Duursma-Lee algorithm, and introduce the supersingular elliptic curves upon which we base our implementation. In Section 4.3 we develop fast arithmetic for pairing values, and in Section 4.5 we give algorithms for efficient exponentiation. In Section 4.4, we describe the field representation we use, while in Section 4.6 we detail our improvements to the Duursma-Lee algorithm. In Section 4.7, we present implementation results, and in the final section, we make some concluding remarks and present some open problems.

4.2 The Modified Tate Pairing

At Asiacrypt 2003, Duursma and Lee introduced an algorithm for pairing computation on a special family of supersingular hyperelliptic curves [30]. In common with the authors of [137], for the elliptic case, which occurs only in characteristic three, we refer to the algorithm as the *modified* Tate pairing.

The modified Tate pairing improves upon the reduced variant in three ways.

Figure 4.1: Field definitions and curve equations

Field	Field Polynomial	Curve	MOV security
$\mathbb{F}_{3^{79}}$	$t^{79}+t^{26}+2$	$Y^2 = X^3 - X - 1$	750
$\mathbb{F}_{3^{97}}$	$t^{97}+t^{12}+2$	$Y^2 = X^3 - X + 1$	906
$\mathbb{F}_{3^{163}}$	$t^{163}+t^{80}+2$	$Y^2 = X^3 - X - 1$	1548
$\mathbb{F}_{3^{193}}$	$t^{193}+t^{12}+2$	$Y^2 = X^3 - X - 1$	1830
$\mathbb{F}_{3^{239}}$	$t^{239}+t^{24}+2$	$Y^2 = X^3 - X - 1$	2268
$\mathbb{F}_{3^{353}}$	$t^{353}+t^{142}+2$	$Y^2 = X^3 - X - 1$	3354

Firstly, using the third property listed in Section 2.4.1, instead of computing the Tate pairing of order l, where l is the order of the subgroup of interest, one uses the pairing of order q^3+1, which eliminates the need for any point additions in Miller's algorithm. Secondly, while this apparently increases the trit-length of the exponent by a factor of three, Duursma and Lee show that the divisor computed when processing three trits at a time has a very simple form, and hence no losses are incurred. Lastly, they provide a closed form expression for the pairing, thus simplifying implementations. We give a full description of the Duursma-Lee algorithm in Section 4.6, where we also make some elementary computational improvements.

The particular curves for which we are interested in optimising the computation of the modified Tate pairing are listed in Figure 4.1 (though this list is clearly not exhaustive). The first column gives the field over which each curve is defined, and the second lists the corresponding irreducible polynomial defining the field extensions. The third lists the curve equation and the final column gives the bit-length of the smallest finite field into which the pairing value embeds, which is a degree six extension for these curves. These parameter values were generated simply by testing which prime extension degrees yielded orders for supersingular curves that are prime, or almost prime, i.e., those possessing a small cofactor, some of which also appear in [42].

4.3 The Quotient Group

Throughout this section and the remainder of the chapter we assume we are working in characteristic three fields with prime extension degree (though the ideas apply equally well to arbitrary finite fields) and so where relevant, all exponents are written in ternary.

Let $l \mid \#E(\mathbb{F}_q)$ and suppose we wish to compute the modified Tate pairing of order l. Then invoking the third property of Section 2.4.1, one uses the Duursma-Lee algorithm to first compute e_{q^3+1}, which is an element in the quotient group

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

For any $a \in \mathbb{F}_{q^6}^{\times}$ we have $a^{q^3+1} \in \mathbb{F}_{q^3}^{\times}$, and so $\mathcal{G}$ simplifies to $\mathbb{F}_{q^6}^{\times}/\mathbb{F}_{q^3}^{\times}$.

Let $G_l \subset \mathbb{F}_{q^6}^{\times}$ denote the subgroup of order l, and let $e \in \mathcal{G}$. Then the two properties

$$\gcd(l, q^3 - 1) = 1 \text{ and } e^{q^3-1} \in G_l$$

imply that $e = gh$ for some $g \in G_l$, $h \in \mathbb{F}_{q^3}^{\times}$. Hence powering e by $q^3 - 1$ gives

$$e^{q^3-1} = (gh)^{q^3-1} = g^{q^3-1},$$

which can then be used in protocols. If a particular protocol requires an exponentiation of this value by some integer $k \bmod l$, this is performed in $\mathbb{F}_{q^6}$.

In this section we give an alternative way to obtain unique representatives of $\mathcal{G}$ easily, that furthermore permits fast multiplication, and provides automatic compression by a factor of two. We then show that the natural embedding of $\mathcal{G}$ into the extension field is just a special representation of an algebraic torus, as given in Section 3.2.3, which also permits further compression, but surprisingly without any further computation.

4.3.1 The Basic Idea

Let $\mathbb{F}_{q^6} = \mathbb{F}_{q^3}[\sigma]/(\sigma^2+1)$ which is the degree two extension we use in the Duursma-Lee algorithm. Writing $e = e_0 + e_1\sigma$ and $g = g_0 + g_1\sigma$, by the above we have

$$e = gh = g_0h + g_1h\sigma.$$

Since the represented coset remains invariant under multiplication by elements of $\mathbb{F}_{q^3}^{\times}$, we can divide by e_1, giving

$$e' = ee_1^{-1} = e_0/e_1 + \sigma = g_0/g_1 + \sigma.$$

This also eliminates h and may equally well be used as a canonical representative of the coset to which e belongs.

This element of the quotient group can be represented simply by the $\mathbb{F}_{q^3}$ element e_0/e_1, and thus compresses the coset representation by a factor of two. Computationally, this involves a division in $\mathbb{F}_{q^3}$.

Comparing this to powering by q^3-1, the saving is not significant, since

$$e^{q^3-1} = \frac{e_0 - e_1\sigma}{e_0 + e_1\sigma},$$

and hence requires only a division in $\mathbb{F}_{q^6}$, which easily reduces to an inversion in $\mathbb{F}_{q^3}$.

However if one exponentiates this value by some integer $k \bmod l$, this operation will be faster than if one had first powered e by q^3-1, since multiplying a generic element of $\mathcal{G}$ by this element is cheaper than multiplying two generic elements, for exactly the same reason as in Section 3.2.3. Letting $g = g_0/g_1 = e_0/e_1$ and $a_0 + a_1\sigma \in \mathcal{G}$, we see that

$$(g+\sigma)(a_0 + a_1\sigma) = (ga_0 - a_1) + (ga_1 + a_0)\sigma,$$

which costs just two $\mathbb{F}_{q^3}$ multiplications, and not the three required if both elements are generic, in which case the arithmetic is identical to that of $\mathbb{F}_{q^6}$. If one assumes cubings and additions are essentially free, then this method will always be roughly one third faster, for whatever practical method one uses to exponentiate. The defining property of the quotient group $\mathcal{G}$ thus reduces the cost of arithmetic performed on pairing values. One can then obtain a canonical representative as before.

4.3.2 Arithmetic in $\mathcal{G}$

We first introduce some terminology to clarify the operations available in $\mathcal{G}$. The property that a given coset is invariant under multiplication by elements of $\mathbb{F}_{q^3}^{\times}$ is suggestive of the projective line

$$\mathbb{P}^1(\mathbb{F}_{q^3}) = \{(x, y) \in (\mathbb{F}_{q^3})^2 \setminus \{(0, 0)\}\}\sim$$

where $(x_1, y_1) \sim (x_2, y_2)$ if and only if a $\lambda \in \mathbb{F}_{q^3}^{\times}$ exists such that $(x_1, y_1) = (\lambda x_2, \lambda y_2)$. The reduction of e to e_0/e_1 may also be viewed as a map to the affine line $\mathbb{A}^1(\mathbb{F}_{q^3})$. With this analogy we introduce the following.

Definition 4.1. *$\mathcal{G}_P$ is the projective line $\mathbb{P}^1(\mathbb{F}_{q^3})$ endowed with the group operation (see Lemma 4.1) induced by the arithmetic of the quadratic extension $\mathbb{F}_{q^6} = \mathbb{F}_{q^3}[\sigma]/(\sigma^2 + 1)$ via the map $(x, y) \to x + y\sigma$. The identity element is represented by the points $(\lambda, 0)$ for any $\lambda \in \mathbb{F}_{q^3}^{\times}$.*

$\mathcal{G}_A$ is the affine part of the line $\mathcal{G}_P$. The affine point corresponding to (x,y) is $X = A(x, y) = (x/y)$. Via this map the identity element is the point at infinity which we denote by $\mathcal{O}_{\mathcal{G}}$.

With this terminology it should be clear that we can mimic mixed addition methods for point multiplication on elliptic curves [23], and it also gives a proper explanation for the representation F_3 in Chapter 3: $\mathcal{G}$ is just $T_2(\mathbb{F}_{q^3})$. It is clear that all the techinques of that chapter transfer directly to this scenario, such as the use of signed digit representations, or exponentiation using a split exponent method, for example (see Section 4.5). Since in this chapter we are interested specifically characteristic three arithmetic, some differences naturally arise.

Let $P = (x, y) \in \mathcal{G}_P$ with corresponding affine representation $(X) \in \mathcal{G}_A$. Again, applying the cube of the Frobenius automorphism gives the inverse of a point, in this case $P^{-1} = (x, -y)$, or $(-X)$ in affine. Cubing is also straightforward since we are working in characteristic three: $P^3 = (x^3, -y^3)$.

For multiplication of two points $P_1 = (x_1, y_1), P_2 = (x_2, y_2) \in \mathcal{G}_P$ with affine representations $(X_1), (X_2) \in \mathcal{G}_A$, we use the following easy lemma.

Lemma 4.1. *Let M and I represent the cost of a multiplication and inversion respectively in $\mathbb{F}_{q^3}$. Then the group operation for combinations of point representations is computed as follows.*

P_1	P_2	$P_1 \cdot P_2$	*Formula*	*Cost*
$\mathcal{G}_A$	$\mathcal{G}_A$	$\mathcal{G}_A$	$(X_1X_2 - 1)/(X_1 + X_2)$	$2M + I$
$\mathcal{G}_A$	$\mathcal{G}_A$	$\mathcal{G}_P$	$(X_1X_2 - 1, X_1 + X_2)$	$1M$
$\mathcal{G}_P$	$\mathcal{G}_P$	$\mathcal{G}_A$	$(x_1x_2 - y_1y_2)/(x_1y_2 + x_2y_1)$	$4M + I$
$\mathcal{G}_P$	$\mathcal{G}_P$	$\mathcal{G}_P$	$(x_1x_2 - y_1y_2, x_1y_2 + x_2y_1)$	$3M$
$\mathcal{G}_A$	$\mathcal{G}_P$	$\mathcal{G}_A$	$(X_1x_2 - y_2)/(X_1y_2 + x_2)$	$3M + I$
$\mathcal{G}_A$	$\mathcal{G}_P$	$\mathcal{G}_P$	$(X_1x_2 - y_2, X_1y_2 + x_2)$	$2M$

Squaring can naturally be performed with slightly fewer $\mathbb{F}_{q^3}$ muliplications than above; the corresponding formulae are easily deduced. Besides the precomputation necessary for the exponentiation algorithms we present in Section 4.5 however, squarings are not required.

With regard to exponentiations, it is clear that the mixed multiplication shown in the final row is the most efficient. If we want to compute P^k for some $k \bmod l$, we first convert P to affine and for each non-zero trit in the expansion of k perform a mixed multiplication of this point with the projective representation of the intermediate value. A multiplication with both points in projective form is equivalent to an ordinary multiplication in $\mathbb{F}_{q^6}$, so the mixed multiplication is essentially what allows the savings over arithmetic in $\mathbb{F}_{q^6}$. We exploit these observations in the exponentiation algorithms detailed in Section 4.5.

4.3.3 Further Compression using $T_6(\mathbb{F}_q)$

Since the characteristic three supersingular elliptic curves we consider have embedding degree six, one may ask why we use the arithmetic of $T_2(\mathbb{F}_{q^3})$ when the order l subgroup is in fact in $T_6(\mathbb{F}_q)$? The reason is that there seems no obvious way to utilise the extra structure provided by $T_6(\mathbb{F}_q)$ [56], though we do not rule out such a possibility. However we know that $|T_6(\mathbb{F}_q)| = (q^2 - q + 1) \mid (q^3 + 1) = |T_2(\mathbb{F}_{q^3})|$, and so $T_6(\mathbb{F}_q) \subset T_2(\mathbb{F}_{q^3})$. Thus one can use the properties of the latter and apply them to the former, utilising the improvements derived over the extension field representation.

While arithmetic improvements do not seem available with T_6, one can exploit it for better compression. As T_6 is rational, one can map nearly all its elements to the affine plane and use this representation instead for data transmissions.

Using the method described by Rubin and Silverberg [127], and thanks to some serendipitous equations for characteristic three and the given field representation, one obtains this additional compression for free.

By Definition 2.5, we know that

$$T_6(\mathbb{F}_q) = \mathrm{Ker}(N_{\mathbb{F}_{q^6}/\mathbb{F}_{q^3}}) \cap \mathrm{Ker}(N_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}) = T_2(\mathbb{F}_{q^3}) \cap \mathrm{Ker}(N_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}).$$

To obtain a suitable representation one therefore only needs to parametrise those elements of $T_2(\mathbb{F}_{q^3})$ which have norm equal to one in the second factor. Let $e = (a - \sigma)/(a + \sigma)$ be the compressed representation for e, and let $a = a_0 + a_1\rho + a_2\rho^2$ where $\rho^3 - \rho \pm 1 = 0$, which defines the cubic extension we later use for the Duursma-Lee algorithm. Then we obtain an equation in a_0, a_1, and a_2 by the condition

$$\left(\frac{a - \sigma}{a + \sigma}\right)^{1+q^2+q^4} = 1.$$

Expanding and simplifying this is equivalent to $1 + a_1^2 - a_0a_2 - a_2^2 = 0$, which one can parametrise easily with just a_1 and a_2, since $a_0 = (1 + a_1^2 - a_2^2)/a_2$. It is therefore sufficient to specify only a_1 and a_2, to describe all points on $T_6(\mathbb{F}_q)$ bar the identity, and this is essentially all that we need. We therefore have a map

$\psi : \mathbb{A}^2(\mathbb{F}_q) \setminus \{(a_1, 0)\} \to T_6(\mathbb{F}_q) \setminus \{1\}$ given by

$$\psi(a_1, a_2) = \frac{((1 + a_1^2 - a_2^2) + a_1a_2\rho + a_2^2\rho^2) - a_2\sigma}{((1 + a_1^2 - a_2^2) + a_1a_2\rho + a_2^2\rho^2) + a_2\sigma}.$$

The inverse map $\psi^{-1} : T_6(\mathbb{F}_q) \setminus \{1\} \to \mathbb{A}^2(\mathbb{F}_q) \setminus \{(a_1, 0)\}$ is given as above, i.e., we just take the second and third coefficients in the fractional expression for e.

Note that $\mathbb{A}^2(\mathbb{F}_q) \setminus \{(a_1, 0)\}$ and $T_6(\mathbb{F}_q) \setminus \{1\}$ both have cardinality $q^2 - q$. In terms of the quotient group $\mathcal{G}$ and an actual pairing computation, once e_0/e_1 is computed, one can therefore use the second and third coefficients to parametrise the element, without any further computation, as mentioned in Section 4.3.

Remark 4.1. *In the context of compression, Rubin and Silverberg [126,128] have shown how one can compress BLS short-signatures [15] by using the trace-zero subvariety contained in the Weil restriction of scalars of an elliptic curve defined over a composite field extension. This method provides a compression factor of* $n/\phi(n)$ *also, where* $\gcd(n, 2) = 1$*, and can be applied to any pairing-based protocol where one is required to transmit a point on the curve, such as [66]. However for* $n \geq 5$*, building upon an idea of Semaev [138], Gaudry has shown that such curves are weaker than those defined over a prime field [47]. Hence this method should be regarded with some caution. We point out that this form of pre-compression is distinct from the post-compression described here, and thus these attacks do not apply.*

4.4 Field Representation

We briefly describe efficient arithmetic for $\mathbb{F}_q$ and the required extensions, before giving several exponentiation methods in Section 4.5.

Field Arithmetic in $\mathbb{F}_q$

Let $\mathbb{F}_q = \mathbb{F}_{3^m}$. Let $a = a_{m-1}x^{m-1} + \cdots + a_1x + a_0$ be an element of $\mathbb{F}_q$, held in a polynomial basis, so that $a_i \in \mathbb{F}_3$. We follow other work [43, 60] and represent the element a as two bit-vectors a_H and a_L. If we let $a_H[i]$ and $a_L[i]$ denote bit i of a_H and a_L respectively, the vectors a_H and a_L are constructed from a such that for all i

$$\begin{aligned} a_H[i] &= a_i \text{ div } 2 \\ a_L[i] &= a_i \bmod 2. \end{aligned}$$

That is, a_H and a_L are a bit-sliced representation of the coefficients of a where a_H holds the high bit and a_L the low bit of a given coefficient. Given a representation of this type, we can perform a component-wise addition $r_i = a_i + b_i$ of two elements a and b using the following word-wise logical operations

$$\begin{aligned} r_H[i] &= (a_L[i] \vee b_L[i]) \oplus t \\ r_L[i] &= (a_H[i] \vee b_H[i]) \oplus t \end{aligned}$$

where

$$t = (a_L[i] \vee b_H[i]) \oplus (a_H[i] \vee b_L[i]).$$

Subtraction, and hence multiplication by two, are equally efficient since the negation of an element a simply swaps the vectors a_H and a_L over and can therefore be implemented by the same function as addition.

On a given computer with word-size w, we hold the bit-vectors a_H and a_L that represent a as two word-vectors of length $n = \lceil m/w \rceil$ and hence apply logical operations in parallel to w coefficients at a time. However, since our representation remains bit-oriented we can borrow further techniques developed for fields of characteristic two. Specifically, it is possible to construct multiplication using a variation of the often cited comb method [95] and inversion by altering the binary extended Euclidean algorithm. We used a Karatsuba method to aggressively split the multiplication operands into word sized chunks, an option that provided significant performance improvements. Unlike elements in characteristic two, squaring in characteristic three is only marginally less expensive than general multiplication. However, cubing can be performed very quickly using table-lookup in an analogous way to the so called *coefficient thinning* method in characteristic two.

Field Arithmetic in $\mathbb{F}_{q^3}$

Let $\mathbb{F}_{q^3} = \mathbb{F}_q[\rho]/(\rho^3 - \rho - b)$, with $b = \pm 1$ depending on the curve equation. Let $a = a_0 + a_1\rho + a_2\rho^2$ and $b = b_0 + b_1\rho + b_2\rho^2$ be two generic elements. We require the following operations.

q-Frobenius:

Since $\rho^3 = \rho + b$ we have $\rho^{3^m} = \rho + (m \bmod 3)b$ and $(\rho^2)^{3^m} = (\rho^{3^m})^2 = \rho^2 + 2b(m \bmod 3)\rho + (m^2 \bmod 3)$. Hence $a^{3^m} = (a_0 + a_1\rho + a_2\rho^2)^{3^m} = (a_0 + a_1 b(m \bmod 3) + a_2 b) + (a_1 - a_2 b(m \bmod 3))\rho + a_2\rho^2$.

Multiplication:

Let $t_{00} = a_0b_0$, $t_{11} = a_1b_1$, $t_{22} = a_2b_2$, $t_{01} = (a_0 + a_1)(b_0 + b_1)$, $t_{12} = (a_1 + a_2)(b_1 + b_2)$, and $t_{20} = (a_2 + a_0)(b_2 + b_0)$. Then $ab = (t_{00} + (t_{12} - t_{11} - t_{22})b) + (t_{01} - t_{00} + t_{11} + t_{12} + t_{22}(b-1))\rho + (t_{20} - t_{00} + t_{11})\rho^2$.

Cubing:

This is straightforward in characteristic three. Since $a^3 = (a_0^3 + a_2^3 + a_1^3 b) + (a_1^3 - a_2^3 b)\rho + a_2^3\rho^2$.

Inversion:

Since the extension degree is small, we can perform this directly. Let $t_{00} = a_0^2$, $t_{11} = a_1^2$, $t_{22} = a_2^2$, $t_{01} = a_0a_1$, $t_{12} = a_1a_2$, $t_{20} = a_2a_0$, and let $\Delta = a_0^3 + a_1^3 b + a_2^3 + t_{20}(a_2 - a_0) - a_1(t_{01} + t_{22}b)$. Then $a^{-1} = \Delta^{-1}((t_{00} - t_{20} + t_{22} - t_{11} - t_{12}b) + (t_{22}b - t_{01})\rho + (t_{11} - t_{20} - t_{22})\rho^2)$.

Field Arithmetic in $\mathbb{F}_{q^6}$

Let $\mathbb{F}_{q^6} = \mathbb{F}_{q^3}[\sigma]/(\sigma^2 + 1)$. Let $c = c_0 + c_1\sigma$ and $d = d_0 + d_1\sigma$ with $c_i, d_i \in \mathbb{F}_{q^3}$ be two generic elements. The arithmetic is as follows.

q-Frobenius:

Since $\sigma^2 = -1$, we have that $\sigma^3 = -\sigma$ and as m is odd, we obtain $c^{3^m} = c_0^{3^m} - c_1^{3^m}\sigma$.

Multiplication:

Let $t_{00} = c_0 d_0$, $t_{11} = c_1 d_1$, and $t_{01} = (c_0 + c_1)(d_0 + d_1)$. Then $cd = (t_{00} - t_{11}) + (t_{01} - t_{00} - t_{11})\sigma$.

Cubing:

$c^3 = c_0^3 - c_1^3\sigma$.

Inversion:

Let $\Delta = c_0^2 + c_1^2$. Then $c^{-1} = \Delta^{-1}(c_0 - c_1\sigma)$.

4.5 Exponentiation

Now that we have set up the basic arithmetic for $\mathcal{G}$, we explore how one can optimise the basic operation of exponentiation in practice. For comparison, we also describe fast algorithms for exponentation in the order l subgroup G_l of $T_6(\mathbb{F}_q)$ using techniques from [148], which is just the representation F_1 of Chapter 3, and also point multiplication in $E(\mathbb{F}_q)$, incorporating a technique similar to the Gallant-Lambert-Vanstone method [44].

For ease of notation we write the group operation for all three groups multiplicatively, and for each of the above we compare four exponentiation methods, which we detail in turn. The input to each algorithm is a base e and an integer $k \bmod l$ in standard ternary format. The output is e^k. When applicable, precomputed values are stored in affine to facilitate the mixed multiplication. Note that in all three groups inversions are essentially for free, so we consider signed digit representations. Note that for the following methods, as stated previously, cubing is fast, and so we consider ternary (and nonary) exponent expanions, rather than binary. Timings are provided for all these methods in Section 4.7.

For the remainder of the chapter, M represents the cost of one $\mathbb{F}_q$ multiplication, rather than a multiplication in $\mathbb{F}_{q^3}$ as in Section 4.3.

Method 1: Signed Ternary Expansion

Using the generalised non-adjacent form, or G-NAF [21], one can take the ternary expansion of an exponent $k \bmod l$ and transform it into an equivalent signed ternary representation. Such a representation is easy to compute and reduces the average density of non-zero trits from two thirds to one half. The precomputation involves just a single squaring of the base.

Method 2: Signed Nonary Expansion

This is the same as Method 1 except we use a base nine expansion of k. This essentially halves the trit-length of k for the cost of precomputing $e^i, i = 1, ..., 8$. Again using the G-NAF, the average density of non-zero 'nits' in this expansion is four fifths.

Method 3: Sliding Window Ternary Expansion

We use an unsigned ternary expansion of k with a sliding window of width three (see Algorithm 14.85 of [101]). To do so one needs to precompute and store e^i for $0 < i < 27$ and $i \neq 0 \bmod 3$.

Method 4: Frobenius Expansion

For $e \in \mathcal{G}$ the q-th power Frobenius automorphism is easily computed. Moreover, the Frobenius of a compressed element is itself compressed. Since the Frobenius map satisfies $q^2 - q + 1 = 0$ (as maps) and the group order divides $q^2 - q + 1$, one can as in Chapter 3 split the exponent k in two halves k_1 and k_2 where k_1, k_2 are approximately half the trit-length of l and satisfy $k \equiv k_1 + k_2 q \mod l$ [148]. One can find k_1 and k_2 very quickly having performed a one-time Gaussian two dimensional lattice basis reduction.

Thus a single exponentiation can be transformed into a double exponentiation for half the trit-length of k, for the cost of performing a double exponentiation instead. To compute e^k for a random $k \bmod l$, we perform the double exponentiation $e^{k_1}(e^q)^{k_2}$ using Shamir's trick, originally due to Straus [150]. We detail the required precomputation in the next section.

For each of k_1, k_2 we invoke the G-NAF. The average density of non-zero trits in each of their ternary expansions is $1/2$ and hence the average number of non-zero trits in the paired ternary expansion of k_1, k_2 is $1 - (1/2)^2 = 3/4$. We therefore expect to perform on average $(3/4) \cdot m/2 = (3/8)m$ multiplications of mixed type during an exponentiation.

This method also works for the elliptic curve, using a method similar to the GLV technique [44] using fast automorphisms. Clearly, one can use the same expansion of k on $E(\mathbb{F}_q)$, replacing powering by q in the field extension with scalar multiplication by q. On the curve, multiplication by q is an efficiently computable automorphism since $[q]P = (x - (m \bmod 3)b, -y)$ for $P = (x, y)$ on the curve (where the curve equation is $Y^2 = X^3 - X + b$).

We note that for supersingular curves over characteristic three there is also an efficient scalar multiplication algorithm due to Koblitz [74] based on the curve automorphism mapping the point (x, y) to (x^3, y^3).

4.5.1 Precomputation

The necessary precomputation for Methods 1, 2 and 3 is straightforward. For Method 4 we can take advantage of the Frobenius map to reduce the cost. For the following we use the notation of $\mathcal{G}$. Let $e = e_0 + e_1\sigma$. In order to use Shamir's trick, we need to know the values

$$(e_0/e_1 + \sigma)^{i+qj} \quad i, j \in \{0, \pm 1, \pm 2\} \tag{4.1}$$

in affine. Let (i, j) represent the corresponding term in (4.1). Then we can use the fact that for any $e \in \mathcal{G}$, we have $e^{q^2-q+1} = \mathcal{O}_{\mathcal{G}}$ to generate most of the required terms easily. To achieve this, one applies the q-Frobenius iteratively to obtain

$(i, j)^q = (-j, i + j)$. We list these operations in Algorithm 1. In G_l we use the same method, having first powered e by $q^3 - 1$, but clearly without needing to obtain affine representatives.

4.5.2 Comparison with Trace-Based Exponentiation

The cost of a mixed multiplication in $\mathcal{G}$ is $12M$. Since $l \approx 3^m$, an exponentiation using Method 4 costs on average about $4.5mM$. This improves considerably on the $12mM$ required by the trace method of [137]. Even without mixed multiplication, this exponentiation still only requires $6.75mM$, and with neither the exponent splitting nor the mixed multiplication, this cost is only about $9mM$. Hence ordinary field arithmetic outperforms the proposed trace method of Scott and Barreto, which in fact can be reduced further to about $10.3mM$ using a Euclidean algorithm [149], but is still over twice as slow.

4.5.3 Application to other Pairings

We have focused primarily on small characteristic tori because the Duursma-Lee algorithm is currently[2] the most efficient for pairing computation. In the future, the preferred embedding degree of a curve will increase in order to maintain a good security/efficiency trade-off, and thus it is likely that ordinary curves over large characteristic fields will be used.

Since the embedding degree n of a pairing on a given abelian variety is minimal, the output of any pairing may be considered an element of the torus T_n. Hence all of the techniques developed for torus-based cryptography may be applied, certainly for any embedding degrees up to thirty (cf. Chapter 5), if such curves can be efficiently found.

However, the results of Chapter 3 show that for large characteristic, the trace-based methods such as LUC [146] (for degree two extensions), and XTR [88, 149] (for degree six extensions), are slightly faster than the torus approach. For the near future however, our methods are likely to remain near-optimal.

[2] See footnote in the chapter introduction.

Algorithm 1: Online Pre-computation for Double Exponentiation

```
input  : $e = e_0 + e_1\sigma \in \mathcal{G}$
output : Representatives in $\mathcal{G}_A$ of

         $(i,j) := (e_0 + e_1\sigma)^{i+jq},\;\; i,j \in \{0, \pm 1, \pm 2\}$

$(1,0) \leftarrow A(e)$
$(0,1) \leftarrow -(1,0)^q$
$(-1,1) \leftarrow -(0,1)^q$
$(-1,0) \leftarrow -(-1,1)^q$
$(0,-1) \leftarrow -(-1,0)^q$
$(1,-1) \leftarrow -(0,-1)^q$

$(2,0) \leftarrow \mathrm{mul}((1,0),(1,0))$
$(2,0) \leftarrow A((2,0))$
$(0,2) \leftarrow -(2,0)^q$
$(-2,2) \leftarrow -(0,2)^q$
$(-2,0) \leftarrow -(-2,2)^q$
$(0,-2) \leftarrow -(-2,0)^q$
$(2,-2) \leftarrow -(0,-2)^q$

$(1,1) \leftarrow \mathrm{mul}((1,0),(0,1))$
$(1,1) \leftarrow A((1,1))$
$(-1,2) \leftarrow -(1,1)^q$
$(-2,1) \leftarrow -(-1,2)^q$
$(-1,-1) \leftarrow -(-2,1)^q$
$(1,-2) \leftarrow -(-1,-1)^q$
$(2,-1) \leftarrow -(1,-2)^q$

$(1,2) \leftarrow \mathrm{mul}((1,0),(0,2))$
$(1,2) \leftarrow A((1,2))$
$(-1,-2) \leftarrow -(1,2)$

$(2,1) \leftarrow \mathrm{mul}((2,0),(0,1))$
$(2,1) \leftarrow A((2,1))$
$(-2,-1) \leftarrow -(2,1)$

$(2,2) \leftarrow \mathrm{mul}((2,0),(0,2))$
$(2,2) \leftarrow A((2,2))$
$(-2,-2) \leftarrow -(2,2)$
```

4.6 Computing the Modified Tate Pairing

In this section we detail how to efficiently implement the Duursma-Lee algorithm for the computation of the modified Tate pairing [30].

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be points of order l on the supersingular curve $E(\mathbb{F}_q) : Y^2 = X^3 - X + b$, for $b \in \{-1, 1\}$. Then the modified Tate pairing on E is the mapping $f_P(\phi(Q))^{q^3-1}$ where $\phi : E(\mathbb{F}_q) \rightarrow E(\mathbb{F}_{q^6})$ is the distortion map $\phi(x_2, y_2) = (\rho - x_2, \sigma y_2)$, where f_P is as defined in Section 2.4.1, and ρ and σ are as defined in Section 4.4. Algorithm 2 gives a closed form expression for this computation, as given by Duursma and Lee, but without the final powering: making use of the techniques of Section 4.3, we do not need to perform this operation, as we presume the output will be stored and transmitted in compressed form.

Algorithm 2: The *Duursma-Lee* Algorithm

```
input    : point $P = (x_1, y_1)$, point $Q = (x_2, y_2)$
output   : $f_P(\phi(Q)) \in \mathcal{G}$

$f \leftarrow 1$
for $i = 1$ to $m$ do
    $x_1 \leftarrow x_1^3, y_1 \leftarrow y_1^3$
    $\mu \leftarrow x_1 + x_2 + b, \lambda \leftarrow -y_1 y_2 \sigma - \mu^2$
    $g \leftarrow \lambda - \mu\rho - \rho^2, f \leftarrow f \cdot g$
    $x_2 \leftarrow x_2^{1/3}, y_2 \leftarrow y_2^{1/3}$
return $f$
```

4.6.1 Cost Analysis

Let M denote the cost of an $\mathbb{F}_q$ multiplication. Each iteration of the loop requires $2M$ to compute μ^2 and $y_1 y_2$, and an $\mathbb{F}_{q^6}$ multiplication to compute $f \cdot g$. Since a generic $\mathbb{F}_{q^6}$ multiplication costs $18M$, Scott and Barreto [137] state that besides the necessary cubings and cube roots, each loop iteration costs $20M$. However, in each iteration g is sparse, i.e., not all of its terms are non-trivial. One can exploit this to reduce the cost of multiplying g and f, which is not sparse in general, to $13M$. This total of $15M$ improves on the trace-based method suggested by Scott and Barreto. In fact one can reduce the cost for each loop iteration in the ordinary Duursma-Lee algorithm to just $14M$, by unrolling the main loop and better exploiting the sparsity of g.

Algorithm 3: A Refined *Duursma-Lee* Algorithm.

```
input   : point $P = (x_1, y_1)$, point $Q = (x_2, y_2)$
output  : $f_P(\phi(Q)) \in \mathcal{G}$

$f \leftarrow 1$
for $i = 1$ to $(m-1)/2$ do
    $x_1 \leftarrow x_1^3, y_1 \leftarrow y_1^3$
    $\mu \leftarrow x_1 + x_2 + b, \lambda \leftarrow -y_1 y_2 \sigma - \mu^2$
    $g_1 \leftarrow \lambda - \mu\rho - \rho^2$
    $x_2 \leftarrow x_2^{1/3}, y_2 \leftarrow y_2^{1/3}$
    $x_1 \leftarrow x_1^3, y_1 \leftarrow y_1^3$
    $\mu \leftarrow x_1 + x_2 + b, \lambda \leftarrow -y_1 y_2 \sigma - \mu^2$
    $g_2 \leftarrow \lambda - \mu\rho - \rho^2$
    $g \leftarrow g_1 g_2, f \leftarrow f \cdot g$
    $x_2 \leftarrow x_2^{1/3}, y_2 \leftarrow y_2^{1/3}$
$x_1 \leftarrow x_1^3, y_1 \leftarrow y_1^3$
$\mu \leftarrow x_1 + x_2 + b, \lambda \leftarrow -y_1 y_2 \sigma - \mu^2$
$g \leftarrow \lambda - \mu\rho - \rho^2, f \leftarrow f \cdot g$
return $f$
```

We demonstrate this technique in Algorithm 3 which provides a saving since in each loop, multiplying g_1 by g_2 costs only $6M$. Multiplying g by f in each loop costs $18M$ since they are both generic $\mathbb{F}_{q^6}$ elements. Both μ^2 and $y_1 y_2$ are computed twice in each loop: once for g_1 and once for g_2. In total the cost therefore is $(6M + 4M)(m-1)/2 + 18M(m-3)/2 + 13M = 14mM - 19M$, which is equivalent to about $14M$ per loop iteration of Algorithm 2.

This analysis ignores the cost of computing cubings and cube roots. Because of the large number of times each of these operations are invoked, it has been suggested that one should use normal bases to accommodate them efficiently, since they are then implemented using cyclic shifts. Normal bases are well-studied in even characteristic, but for characteristic three one can not construct optimal, type one normal bases with prime extension degree [45, 111], although type two bases are available for some values of m. As a result, the cost of general multiplication in software is relatively large, even when variations of high performance methods in characteristic two are used [110, 123]. For example, we found that when $m = 239$ normal basis multiplication is between two and three times slower than a polynomial basis multiplication [58]. However, in hardware implementations on a smart-card for example, normal bases still seem the obvious choice since they can match the multiplication speed of polynomial basis while offering inexpensive cube and cube root operations, although perhaps at the cost of flexibility.

To reduce the cost of computing cube roots using a polynomial basis, we observe that the successive cube roots of x_2 and y_2 can be computed more easily in reverse order and stored for the duration of the algorithm. Since for any $x_2 \in \mathbb{F}_q$, we have $x_2 = x_2^{3^m}$, the required values $x_2^{1/3^i}$ can be computed as $x_2^{3^{m-i}}$, and thus one does not need to compute any cube roots at all. The memory requirement for this is only about $2^{-11}m^2$ Kb and the time taken is just the cost of $2m$ cubings. If memory is at a premium, one can reduce this to about $2^{-4.5}m^{3/2}$ Kb with double the number of cubings using further loop unrolling and pebbling strategies.

Remark 4.2. *As already mentioned, Scott and Barreto's method [137] is effectively a change of basis and not a compressed method of computing a pairing. Hence it is unsurprising that the loop unrolling strategy of Algorithm 3 can be used to reduce the cost of the trace method given there, as kindly pointed out by Barreto [5].*

Remark 4.3. *Scott and Barreto [137] also suggested an open problem asking if it is possible to perform the pairing computation directly in compressed form for some compression factor ≥ 3 on ordinary (non-supersingular) curves in characteristic $p > 3$. A compression factor larger than 3 is extremely unlikely. For pairing-based applications, the desirable extension degrees in the near future are likely to remain small, and no larger than twenty. By Lemma 1, the maximum compression factor possible for a given extension degree n is $n/\phi(n)$, and for $n < 20$, this maximum is three, which is already achieved for the modified Tate pairing.*

Figure 4.2: Pairing and exponentiation timings (ms).

	$\mathbb{F}_{3^{79}}$	$\mathbb{F}_{3^{97}}$	$\mathbb{F}_{3^{163}}$	$\mathbb{F}_{3^{193}}$	$\mathbb{F}_{3^{239}}$	$\mathbb{F}_{3^{353}}$
Pairing						
BKLS	13.96	23.60	79.11	123.21	179.30	527.56
Algorithm 3	4.67	8.41	29.26	45.67	65.73	197.58
Exponentiation in G_l						
Method 1	3.65	6.14	20.98	33.21	44.72	130.27
Method 2	4.57	7.25	21.53	31.61	43.56	119.16
Method 3	3.67	5.79	17.85	26.69	36.45	101.75
Method 4	3.06	5.10	16.55	24.67	34.74	99.56
Exponentiation in $\mathcal{G}$						
Method 1	2.55	4.27	14.15	21.67	30.69	88.06
Method 2	2.62	5.21	13.21	20.38	26.97	74.90
Method 3	3.69	4.72	15.78	22.96	37.96	73.29
Method 4	2.32	4.07	11.84	17.63	24.73	69.30
Point Multiplication in $E(\mathbb{F}_q)$						
Method 1	1.83	3.11	10.62	16.94	24.11	69.78
Method 2	1.72	2.84	9.47	14.73	21.15	60.70
Method 3	1.82	3.01	9.66	14.95	21.19	58.70
Method 4	1.18	1.95	8.11	12.75	19.04	55.93

4.7 Implementation Results

In order to provide some concrete idea of the practical cost of these and other methods, the proposed field arithmetic, pairing algorithms and exponentiation methods were implemented. A GCC 3.3 compiler suite was used to build the implementation, which was run on a Linux based PC incorporating a 2.80 GHz Intel Pentium 4 processor to perform timing experiments. The entire system was constructed in C++. While it would be possible to make further performance improvements through aggressive profiling and optimisation, these results are intended to be representative of the underlying algorithms and to allow a fair comparison between them.

Figure 4.2 shows the result of timing this implementation using a variety of different base field sizes. In the pairing section, Algorithm 3 refers to the aug-

Figure 4.3: Field operation timings (μs).

	$\mathbb{F}_{3^{79}}$	$\mathbb{F}_{3^{97}}$	$\mathbb{F}_{3^{163}}$	$\mathbb{F}_{3^{193}}$	$\mathbb{F}_{3^{239}}$	$\mathbb{F}_{3^{353}}$
$\mathbb{F}_q$						
Add	0.55	0.53	0.58	0.63	0.61	0.64
Square	4.42	6.07	12.99	16.48	19.48	40.97
Cube	0.85	0.84	0.96	1.26	1.24	1.77
Invert	23.18	33.26	70.10	97.20	136.86	303.27
Multiply	4.06	6.02	12.80	17.83	19.42	43.11
$\mathbb{F}_{q^3}$						
Add	0.60	0.60	0.80	0.90	0.90	0.50
Cube	2.10	2.10	2.30	2.50	3.20	4.20
Invert	65.00	94.70	204.40	275.90	350.60	741.80
Frobenius	1.10	0.90	1.10	1.00	1.30	1.40
Multiply	26.10	37.80	74.20	98.00	115.50	249.00
$\mathbb{F}_{q^6}$						
Add	0.90	0.90	0.90	1.10	1.00	1.10
Cube	2.80	4.60	4.40	4.00	5.00	5.60
Invert	165.50	237.20	497.40	670.10	817.10	1709.50
Frobenius	2.00	2.10	1.90	2.00	2.10	2.10
Multiply	75.70	106.10	227.10	296.80	347.30	745.10

mented version of Duursma-Lee presented in this paper, with the cube root pre-computation strategy and the loop unrolling. The BKLS method is included as a reference. We do not include timings for the methods of [137] since our operation count clearly shows they will be slower than our alternatives. Figure 4.3 gives timings for the underlying field operations.

We note first that the implementation of Algorithm 3 is between two to three times faster than the BKLS algorithm. With regard to exponentiation, Method 4 is the most efficient for all field sizes and in all three groups, and in $\mathcal{G}$ is nearly twice as fast as Method 1 in G_l. An early estimate of Koblitz [74] states that the ratio of the time required for an exponentation in $\mathbb{F}_{q^6}$ to the time required for a point multiplication in $E(\mathbb{F}_q)$ is 12; these results demonstrate that for fields of a cryptographic size, this value is in fact closer to 1.3. Thus the techniques from [148], together with the fast multiplication in $\mathcal{G}$, improve the efficiency of post-pairing arithmetic considerably.

We conceed that while Koblitz's complex multiplication exponentiation method has not been implemented, due to the estimated large preprocessing time required, we do not think it would affect this comparison significantly.

Furthermore, due to our direct inversion method, the ratio of inversion time to multiplication time in $\mathbb{F}_{q^3}$ is under three for all field sizes. This means our compression method in $\mathcal{G}$ costs roughly $4/3$ multiplications in $\mathbb{F}_{q^6}$, and is therefore also very efficient.

4.8 Conclusion and Open Problems

In this chapter we have shown how to take advantage of the quotient group to which a pairing value naturally belongs in order to speed up exponentiations, and to obtain fast compression of pairing values. We have also proposed some simple refinements to the Duursma-Lee algorithm to improve efficiency. Our results strongly indicate that there are definite advantages to implementing pairing-based cryptographic protocols in characteristic three: the often quoted value of ten for the ratio of the speed of a pairing evaluation to a point multiplication on the curve is really closer to three or four.

Some issues remain. One could certainly improve the exponentiation times for all three groups if there exists an efficiently computable ternary analogue of the Joint Sparse Form [147]. With regard to side channel attacks, such a method may be undesirable since one can not render cubing and multiplication in charcteristic three fields indistinguishable without a serious detriment to performance. As such, a cube-and-multiply-always method using the exponent splitting of Method 4 will half the cost of a secure full length expansion.

Also the exact security of the discrete logarithm problem in characateristic three using the ternary analogue of Coppersmith's method has yet to be investigated [24, 25]. Preliminary research into this problem using Adleman's Function Field Sieve has been conducted - see Chapters 7 and 8 - but the problem should still be considered open.

Lastly, do there exist methods for faster pairing evaluation using so-called MNT curves [104], which form a family of non-supersingular elliptic curves over large prime fields with embedding degree six also? Work by Page, Smart and Vercauteren [118] indicates that which method works best depends on the application. Recently however, special methods developed for supersingular curves [6, 30, 80], have to some extent been adapted to ordinary elliptic curves, see [63].

Chapter 5

Practical Cryptography in High Dimensional Tori

In this chapter we present practical and efficient methods for the compression of sequences of elements of arbitrary algebraic tori, which are asymptotically optimal with the number of elements to be transmitted.

This chapter represents joint work with Marten van Dijk, Dan Page, Karl Rubin, Alice Silverberg, Martijn Stam and David Woodruff, and appeared in [156].

5.1 Introduction

At Crypto 2004, van Dijk and Woodruff introduced a new way of using the algebraic tori T_n in cryptography, and obtained an asymptotically optimal $n/\phi(n)$ savings in bandwidth and storage for a number of cryptographic applications. However, the computational requirements of compression and decompression in their scheme were impractical, and it was left open to reduce them to a practical level. We give a new method that compresses orders of magnitude faster than the original, while also speeding up the decompression and improving on the compression factor (by a constant term). Further, we give the first efficient implementation that uses T_{30}, compare its performance to CEILIDH, XTR and ECC, and present new applications. Our methods achieve better compression than CEILIDH and XTR for the compression of as few as two group elements. This allows us to apply our results to ElGamal encryption with a small message domain to obtain ciphertexts that are 10% smaller than in previous schemes.

Although T_n is not known to be rational in general, van Dijk and Woodruff [157] show that one can obtain key agreement, signature and encryption schemes with a compression factor asymptotically $n/\phi(n)$ as the number of keys, signatures, or messages grows, *without* relying on the rationality of T_n. The key property of tori they use is that T_n is *stably rational* [161], i.e., for every n there is an m such that there is an "almost bijection"[1] between $T_n(\mathbb{F}_q) \times \mathbb{F}_q^m$ and $\mathbb{F}_q^{\phi(n)+m}$.

Using the fact that T_n is stably rational, van Dijk and Woodruff [157] developed bijections between $T_n(\mathbb{F}_q) \times \mathbb{F}_q^m$ and $\mathbb{F}_q^{\phi(n)+m}$ with $m = \sum_{d|n,\ \mu(n/d)=-1} d$, where μ is the Möbius function, leading to asymptotically optimal $n/\phi(n)$ savings in bandwidth and storage. However, a major drawback of their solution is its large computational requirements.

The present chapter gives a new and efficient construction of bijections between $T_n(\mathbb{F}_q) \times \mathbb{F}_q^m$ and $\mathbb{F}_q^{\phi(n)+m}$ with significantly smaller m than in [157], as well as an optimised implementation when $n = 30$. The latter builds upon the techniques developed in Chapter 3 to efficiently implement CEILIDH.

Note that $n = 30 = 2 \cdot 3 \cdot 5$ is the next cryptographically interesting case, since its compression is up to 20% better than that of systems based on $n = 6$. In addition to our computational savings, in this case we are able to reduce the original affine surplus $m = 32$ [157] to $m = 2$. As we show, this reduction has immediate practical implications.

Since we are interested in the practicality of our construction, we perform timings for exponentiations, compression and decompression for both the new $T_{30}(\mathbb{F}_q)$ system and for a CEILIDH-based $T_6(\mathbb{F}_{q_L})$ system with $q_L \approx q^5$. For the equivalent of 1024-bit RSA security, the computational costs of the operations in both systems are comparable, while the compression of our scheme is better by a factor of $5/4 = (30/\phi(30))/(6/\phi(6))$.

The chapter is organised as follows. In Section 5.2 we describe the central idea

[1]The maps may be undefi ned on a small number of points.

behind the compression method of van Dijk and Woodruff [157]. In Section 5.3 we present our new mapping, and in Section 5.4 give some cryptographic applications. In Section 5.5 we show how to implement our mapping, and in Section 5.6 we present implementation results.

5.2 Asymptotically Optimal Torus-Based Cryptography

Since T_n is known to be rational only for special values of n, one can not use rationality to achieve a compression factor greater than three for a single torus element. Van Dijk and Woodruff [157] overcome this problem in the case where several elements of T_n are to be compressed. They construct a bijection:

$$\theta : T_n(\mathbb{F}_q) \times (\times_{d|n,\mu(n/d)=-1}\mathbb{F}_{q^d}^{\times}) \to \times_{d|n,\mu(n/d)=1}\mathbb{F}_{q^d}^{\times}. \tag{5.1}$$

Specializing their map to the case $n = 30$ gives

$$T_{30}(\mathbb{F}_q) \times \mathbb{F}_q^{\times} \times \mathbb{F}_{q^6}^{\times} \times \mathbb{F}_{q^{10}}^{\times} \times \mathbb{F}_{q^{15}}^{\times} \to \mathbb{F}_{q^2}^{\times} \times \mathbb{F}_{q^3}^{\times} \times \mathbb{F}_{q^5}^{\times} \times \mathbb{F}_{q^{30}}^{\times},$$

which can be reinterpreted as an “almost bijection” (see [157])

$$T_{30}(\mathbb{F}_q) \times \mathbb{A}^{32}(\mathbb{F}_q) \to \mathbb{A}^{40}(\mathbb{F}_q).$$

One can use this map to achieve an asymptotic compression factor of $30/8$. Indeed, to compress m elements of $T_{30}(\mathbb{F}_q)$, one can compress an element x and split its image into $y_1 \in \mathbb{A}^8(\mathbb{F}_q)$ and $y_2 \in \mathbb{A}^{32}(\mathbb{F}_q)$. Then y_1 forms the affine input of the next compression. In the end, $8m + 32$ elements of $\mathbb{F}_q$ are used to represent m elements of $T_{30}(\mathbb{F}_q)$. Observe that their map comes from the equation

$$\Phi_{30}(x)(x-1)(x^6-1)(x^{10}-1)(x^{15}-1) = (x^2-1)(x^3-1)(x^5-1)(x^{30}-1), \tag{5.2}$$

relating the orders of all the different component groups of domain and range. Since these groups are cyclic, one can map to and from their products as long as the orders of the component groups are coprime. For the map above there are some small primes that occur in the order of several component groups, but van Dijk and Woodruff are able to isolate and handle them separately.

5.3 The New Construction

The bijection (5.1), while asymptotically optimal, leaves open the question of whether one can obtain better compression for a *fixed* number of elements. Our new compression map, given by (5.4) below (see Theorems 5.1 and 5.3), has this property. Using the fact that $\Phi_n(x) = \prod_{d|n}(x^d - 1)^{\mu(n/d)}$, we have

Proposition 5.1. *If p is a prime, and a is a positive integer not divisible by p, then*

$$\Phi_{ap}(x)\Phi_a(x) = \Phi_a(x^p).$$

The following result can be deduced from Proposition 5.1 above, using Lemma 6 of [157] (see also pp. 60–61 of [161]). Here, Res denotes the Weil restriction of scalars (see for example [161] or [125]).

Theorem 5.1. *If p is a prime, q is a prime power, a is a positive integer, qa is not divisible by p, and $\gcd(\Phi_{ap}(q), \Phi_a(q)) = 1$, then*

$$T_{ap}(\mathbb{F}_q) \times T_a(\mathbb{F}_q) \cong (\mathrm{Res}_{\mathbb{F}_{q^p}/\mathbb{F}_q} T_a)(\mathbb{F}_q) \cong T_a(\mathbb{F}_{q^p}).$$

The next result follows from Proposition 5.1, by doing double induction on the number of prime divisors of n and the number of prime divisors of m.

Theorem 5.2. *If n is square-free and m is a divisor of n, then*

$$\Phi_n(x) \prod_{d|\frac{n}{m},\, \mu(\frac{n}{md})=-1} \Phi_m(x^d) = \prod_{d|\frac{n}{m},\, \mu(\frac{n}{md})=1} \Phi_m(x^d).$$

The next result follows from Theorem 5.2, using the ideas in the proof of Theorem 3 of [157].

Theorem 5.3. *If n is square-free and m is a divisor of n, then there is an efficiently computable bijection (with an efficiently computable inverse)*

$$T_n(\mathbb{F}_q) \times \prod_{d | \frac{n}{m},\, \mu(\frac{n}{md})=-1} T_m(\mathbb{F}_{q^d}) \to \prod_{d | \frac{n}{m},\, \mu(\frac{n}{md})=1} T_m(\mathbb{F}_{q^d}).$$

Note that [157] is based on the case $m = 1$ of Theorem 5.3. Theorem 5.3 is most useful to us when T_m is rational. If T_m is rational, then Theorem 5.3 gives efficiently computable "almost bijections" between T_m and $\mathbb{A}^{\phi(m)}$, and we have

$$T_n \times \mathbb{A}^{D(m,n)} \sim \mathbb{A}^{\phi(n)+D(m,n)} \tag{5.3}$$

where

$$D(m, n) = \phi(m) \sum_{d | \frac{n}{m},\, \mu(\frac{n}{md})=-1} d$$

and $\sim$ denotes efficient "almost bijections". The smaller $D(m, n)$ is, the better for our applications. Given the current state of knowledge about the rationality of the tori T_m, we take m with at most two prime factors. Ideally, $m = 6$. One could also take $m = 2$. When $m = 6$, then (5.3) gives

$$T_{30} \times \mathbb{A}^2 \sim \mathbb{A}^{10} \qquad \text{and} \qquad T_{210} \times \mathbb{A}^{24} \sim \mathbb{A}^{72}.$$

As a comparison with the original bijection (5.1) for $n = 30$ which requires $8m + 32$ elements of $\mathbb{F}_q$ to represent m elements in $T_{30}(\mathbb{F}_q)$, we see that this provides a considerable improvement.

Even better, using Proposition 5.1 and induction on the number of prime divisors of n, we also obtain the following.

Theorem 5.4. *If $n = p_1 \cdots p_k$ is a product of $k \geq 2$ distinct primes, then*

$$\Phi_n(x) \prod_{i=2}^{k-1} \Phi_{p_1 \cdots p_i}(x^{p_{i+2} \cdots p_k}) = \Phi_{p_1 p_2}(x^{p_3 \cdots p_k}).$$

Applying this to $n = 210 = 2 \cdot 3 \cdot 5 \cdot 7$, one can similarly show

$$T_{210}(\mathbb{F}_q) \times T_{30}(\mathbb{F}_q) \times T_6(\mathbb{F}_{q^7}) \sim T_6(\mathbb{F}_{q^{35}}).$$

Now since $T_6 \sim \mathbb{A}^2$, we obtain $T_{210} \times T_{30} \times \mathbb{A}^{14} \sim \mathbb{A}^{70}$. Using $T_{30} \times \mathbb{A}^2 \sim \mathbb{A}^{10}$ now gives $T_{210} \times \mathbb{A}^{22} \sim T_{210} \times \mathbb{A}^{10} \times \mathbb{A}^{12} \sim T_{210} \times (T_{30} \times \mathbb{A}^2) \times \mathbb{A}^{12} \sim T_{210} \times T_{30} \times \mathbb{A}^{14} \sim \mathbb{A}^{70}$, so

$$T_{210} \times \mathbb{A}^{22} \sim \mathbb{A}^{70}.$$

More generally, the above reasoning shows that if $n = p_1 \cdots p_k$ (square-free), then

$$T_n \times \mathbb{A}^{\phi(p_1 p_2) p_3 \cdots p_n - \phi(n)} \sim \mathbb{A}^{\phi(p_1 p_2) p_3 \cdots p_n},$$

which for $6 \mid n$ gives

$$T_n \times \mathbb{A}^{n/3 - \phi(n)} \sim \mathbb{A}^{n/3}. \tag{5.4}$$

Using (5.4), one can compress m elements of $T_n(\mathbb{F}_q)$ down to just $(m-1)\phi(n) + n/3$ elements of $\mathbb{F}_q$ by either sequential or tree-based chaining as explained in Section 5.4.

5.3.1 Applying the Construction to T_{30}

Henceforth we focus on $n = 30$ since this improves upon previous schemes, has a straightforward parameter generation (see Section 5), and will be computationally efficient. Note that $\gcd(\Phi_{30}(q), \Phi_6(q)) = 1$. Indeed, using the first paragraph of the proof of Lemma 6 of [157], the only possible prime dividing $\gcd(\Phi_{30}(q), \Phi_6(q))$ is 5, but it is easy to see that regardless of q we have $\Phi_6(q) \bmod 5 \in \{1, 2, 3\}$, which proves our claim. By Theorem 5.1 we now have

$$T_{30}(\mathbb{F}_q) \times T_6(\mathbb{F}_q) \cong T_6(\mathbb{F}_{q^5}).$$

The compression is based on a sequence of maps

$$T_{30}(\mathbb{F}_q) \times (\mathbb{A}^2(\mathbb{F}_q) \setminus V(f)) \rightarrow T_{30}(\mathbb{F}_q) \times T_6(\mathbb{F}_q) \rightarrow T_6(\mathbb{F}_{q^5}) \rightarrow \mathbb{A}^2(\mathbb{F}_{q^5}) \setminus V(f_5),$$

where $V(f_5)$ denotes $V(f)$ over $\mathbb{F}_{q^5}$. We denote by θ the forward composition of the three maps above, and by θ^{-1} the composition of the inverses. Note that if we have m elements in $T_{30}(\mathbb{F}_q)$, we compress them down to $8m + 2$ elements of $\mathbb{F}_q$. Thus the compression outperforms CEILIDH and XTR when as few as two elements are compressed.

The first and last maps are based on CEILIDH decompression and compression, respectively. We discuss some possibilities for the map

$$\sigma : T_{30}(\mathbb{F}_q) \times T_6(\mathbb{F}_q) \longrightarrow T_6(\mathbb{F}_{q^5})$$

in Section 5.5 below.

5.3.2 Missing Points

With regard to the functionality of θ, the only remaining issue is that the outer two maps based on CEILIDH do not give everywhere-defined injections.

We can slightly modify the CEILIDH maps, so that for compression we get an injection $\psi' : \mathbb{A}^2(\mathbb{F}_q) \to T_6(\mathbb{F}_q) \times \{0, 1\}$ and for decompression an injection $\rho' : T_6(\mathbb{F}_q) \times \{0, 1\} \to \mathbb{A}^2(\mathbb{F}_q)$. Note that ψ' and ρ' need not be inverses. The two missing points in ρ's domain can easily be added by using a table lookup into two arbitrarily chosen points in $V(f)$. The resulting map is ρ'.

Given the different cardinalities of $T_6(\mathbb{F}_q)$ (namely $q^2 - q + 1$) and $\mathbb{A}^2(\mathbb{F}_q)$ (namely q^2), there are certain points in $\mathbb{A}^2(\mathbb{F}_q)$ that do not decompress. If we concentrate on the case $q \equiv 2 \bmod 9$ or $q \equiv 5 \bmod 9$, then the variety $V(f)$ is defined by $f(v_1, v_2) = 1 - v_1^2 - v_2^2 + v_1 v_2$. For fixed v_2 this has at most 2 roots, and if this is the case then their difference is $(4 - 3v_2^2)^{1/2}$. If this expression equals 2 then $v_2 = 0$, in which case $v_1 \in \{-1, 1\}$. Thus we have a map $\psi' : \mathbb{A}^2(\mathbb{F}_q) \to T_6(\mathbb{F}_q) \times \{0, 1\}$:

- If $f(v_1, v_2) \neq 0$, then $\psi'(v_1, v_2) = (\psi(v_1, v_2), 0)$,
- Else if $v_2 \neq 0$, then $\psi'(v_1, v_2) = (\psi(v_1 + 2, v_2), 1)$,
- Else $\psi'(v_1, v_2) = (\psi(v_1 + 1, v_2), 1)$,

where the extra bit indicates whether the input landed in the variety.

5.4 Applications

Our new map saves a significant amount of communication in applications where many group elements are transmitted. For instance the compression can be used to agree on a sequence of keys using Diffie-Hellman as in Section 5.1 of [157]. Other applications include verifiable secret sharing, electronic voting and mix-nets, and private information retrieval.

In our applications we compress many elements. This is done by using part of the output of the i-th element as the affine input for the compression of the $(i+1)$-st element. This sequential chaining is simple, but has the drawback of needing to decompress all elements in order to obtain the first element. Alternatively, one can use trees to allocate the output of previous compressions. For instance, the output of the first compression is split into five pieces, which are subsequently used as the affine input when compressing elements two through six. The output of the second compression is used to compress elements seven through twelve, etc. When compressing m elements, decompressing a specific element now takes $O(\log m)$ atomic decompressions on average.

5.4.1 ElGamal Encryption

Our first application is ElGamal encryption with a small message domain, where we obtain an additional 10% compression over CEILIDH even for the encryption of a single message, since in ElGamal one transmits two elements. This contrasts starkly with the original mapping of [157] that cannot be used to achieve any savings for single-message encryption.

Let q and l be primes such that $l \mid \Phi_{30}(q)$. Let $g \in \mathbb{F}_{q^{30}}^{\times}$ have order l, so that $\langle g \rangle \subseteq T_{30}(\mathbb{F}_q)$. For random a, $1 \leq a \leq l-1$, let a be Bob's private key and $A = g^a$ his public key. Without loss of generality, let $\mathcal{M} = \{0, 1, \ldots, m-1\}$ be the set of possible messages. We assume that m, the cardinality of $\mathcal{M}$, is small. We apply the mapping of Section 5.3 to the generalized ElGamal encryption scheme.

Encryption (M):

1. Alice represents the message M as $g^M \in \langle g \rangle$.
2. Alice selects a random integer k, $1 \leq k \leq l$, and computes $d = g^k$.
3. Alice sets $e = g^M \cdot (g^a)^k$.
4. Alice expresses $d \in T_6(\mathbb{F}_{q^5})$ as $(d_1, d_2) \in \mathbb{A}^8(\mathbb{F}_q) \times \mathbb{A}^2(\mathbb{F}_q) \cong \mathbb{A}^2(\mathbb{F}_{q^5})$ by using CEILIDH. Alice compresses $e \in T_{30}(\mathbb{F}_q)$ and $d_2 \in \mathbb{A}^2(\mathbb{F}_q)$ as $\theta(e, d_2) = T$, and outputs (d_1, T).

Decryption (d_1, T):

1. Bob computes $\theta^{-1}(T) = (e, d_2)$ and uses CEILIDH to reconstruct d.
2. Bob uses his private key a to recover $g^M = d^{-a}e$.
3. Bob recovers M from g^M using the fact that M comes from a small domain (e.g., using Pollard lambda or a table lookup).

The ciphertext is represented as 18 symbols in $\mathbb{F}_q$, which is a 10% improvement over a solution in which CEILIDH is used to compress both d and e. Note that the mapping of [157] in Section 5.2 cannot be used to achieve any savings in this case.

Our scheme preserves homomorphy, that is, without knowing the secret key a one can compute the encryption of $M_1 + M_2$ given encryptions of M_1 and M_2 separately. This is useful in applications such as the efficient two-party computations proposed by Schoenmakers and Tuyls [135] which use homomorphic ElGamal encryption for a small number of messages.

Exactly as for XTR and CEILIDH (with 6 replaced by 30), the security of our schemes follows from the difficulty of the DDH problem in $\mathbb{F}_{q^{30}}^\times$, the fact that $T_{30}(\mathbb{F}_q)$ is the primitive subgroup of $\mathbb{F}_{q^{30}}^\times$, and the fact that our map and its inverse are efficiently computable.

The representation of M as g^M in $\langle g \rangle$ is not efficient when m is large. There seems no obvious way to circumvent the issue of efficiently encoding m as an element of T_{30}. We thus leave it as an open problem to adapt our scheme to handle a larger message domain. We note that one solution is to use hybrid ElGamal [2] encryption instead. Indeed, we may apply the mapping of Section 5.3 to hybrid ElGamal encryption, adapting a protocol in Section 5.3 of [157]. In general, though, this solution does not preserve the homomorphic property.

5.4.2 ElGamal Signatures

We apply the mapping of Section 5.3 to the generalized ElGamal signature scheme, adapting a protocol in Section 5.2 of [157]. Here the signature has the form (d, e), where $d \in \langle g \rangle$ and $1 \leq e \leq l - 1$. The idea is to use part of e in the affine component of θ, which can be done without any loss since $\log_2 e \approx 160$ while $2 \log_2 q \approx 70$; see Section 5.5.5 for a discussion of parameters. Since the affine component of [157] is much larger, this is not possible in their setting.

For a random a, $1 \leq a \leq l - 1$, let a be Alice's private key and $A = g^a$ her public key. Let $h : \{0,1\}^* \to \mathbb{Z}_l$ be a cryptographic hash function. We have the following generalized ElGamal signature scheme for input message M:

Signature Generation (M):

1. Alice selects a random secret integer k, $1 \leq k \leq l$, and computes $d = g^k$.
2. Alice then computes $e = k^{-1}(h(M) - ah(d)) \bmod l$.
3. Alice expresses e as $(e_1, e_2) \in \mathbb{F}_q^2 \times \{0,1\}^*$, computes $\theta(d, e_1) = T$, and outputs (e_2, M, T) as her signature.

Signature Verification (e_2, M, T):

1. Bob computes $\theta^{-1}(T) = (d, e_1)$ and recovers e.

[2] A hybrid encryption scheme uses public key encryption to encrypt a random symmetric key, and then encrypts the message using the symmetric key, thus overcoming the problem of comparatively slow public key encryption methods.

2. Bob accepts the signature if and only if $A^{h(d)}d^e = g^{h(M)}$.

We note that in practice one has the alternative of using Schnorr's signature scheme, which already achieves optimal compression.

5.5 Representations and Algorithms for T_{30}

In this section we discuss implementation issues concerning field representation, key generation, and efficient exponentiation.

5.5.1 Field Representations

Since $T_{30}(\mathbb{F}_q) \subset \mathbb{F}_{q^{30}}^{\times}$, we need a model of the latter that permits fast multiplication, squaring, inversion and a fast Frobenius automorphism. We also require arithmetic for T_6, over both $\mathbb{F}_q$ and $\mathbb{F}_{q^5}$. Since $T_{30}(\mathbb{F}_q) \subset T_6(\mathbb{F}_{q^5})$, we may model the arithmetic of $T_{30}(\mathbb{F}_q)$ by the latter, possibly at the risk of losing some optimizations.

The base field $\mathbb{F}_q$

We base our implementation on high performance arithmetic in $\mathbb{F}_q$ using the representational method of Montgomery [105]. For T_{30} one is likely to use characteristics q between 32 and 64 bits long, corresponding to a 2-word value on a 32-bit architecture. We are careful to distinguish between those small, 2-word values required by $T_6(\mathbb{F}_{q^5})$ and more general values of q (which we need for comparison purposes). Essentially, we employ the trivial program specialisation techniques described by Avanzi [4] to construct compact, straight line code sequences for the 2-word case. This affords a significant improvement over code for general sizes of q. Other than the length, we do not rely on assumptions on the value of q, although one could expect incremental improvements by doing so. Also, our choice of extension degree poses some congruence conditions on q.

The extension $\mathbb{F}_{q^5}$

We use a degree five subfield of the degree 10 extension $\mathbb{F}_q[t]/(\Phi_{11}(t))$, and use the Gaussian normal basis $\{t + t^{10}, t^7 + t^4, t^5 + t^6, t^2 + t^9, t^3 + t^8\}$. For this to work we require that $q \neq \pm 1 \mod 11$ [111]. Since the extension degree is small, we perform inversions using the Itoh-Tsujii algorithm [65].

The torus T_6

For the torus T_6 we take $q \equiv 2 \mod 9$ or $q \equiv 5 \mod 9$ and use arithmetic based on the degree six extension field defined by adjoining a primitive ninth root of unity to the base field, as in [127, 148] and Chapter 3. Note that in T_6 we have virtually free inversion, as it is just the cube of the Frobenius automorphism.

5.5.2 Compression and Decompression

Our new compression and decompression algorithms require two components: CEILIDH and the Chinese Remainder Theorem. We use the implementation of CEILIDH as given in Chapter 3.

Although it seems that Chinese remaindering is straightforward, there is some flexibility in choosing the map $\sigma : T_{30}(\mathbb{F}_q) \times T_6(\mathbb{F}_q) \to T_6(\mathbb{F}_{q^5})$. Following [157] we have $\sigma(x, y) = x^\beta y^\alpha$, where $\alpha\Phi_{30}(q) + \beta\Phi_6(q) = 1$. The inverse is given by $\sigma^{-1}(z) = (z^{\Phi_6(q)}, z^{\Phi_{30}(q)})$. The cost of the forward computation (i.e., σ) is an exponentiation in $T_{30}(\mathbb{F}_q)$, an exponentiation in $T_6(\mathbb{F}_q)$, and a multiplication. Depending on the context, the exponentiation in $T_{30}(\mathbb{F}_q)$ may be combined with an exponentiation performed as part of a particular protocol. The inverse is a double exponentiation.

Also attractive is the simple $\sigma'(x, y) = xy$ with inverse $(\sigma')^{-1}(z) = (zy^{-1}, y)$ where $y = z^{\alpha\Phi_{30}(q)}$. Clearly the forward map only costs a multiplication. For the inverse we first compute y using a single exponentiation. Note that the exponent here is larger than in the case of σ, but the total amount of exponent is similar in both cases (although asymptotically it is not the total amount that counts, it is what is relevant in practice). Moreover, we are typically concerned with the case

	$\mathbb{F}_{q^5}$	$T_6(\mathbb{F}_q)$	$T_6(\mathbb{F}_{q^5})$
Multiply	$15M + 75A$	$18M + 53A$	$270M + 1615A$
Square		$6M + 21A$	$90M + 555A$
Inverse	$65M + 300A + I$	$2A$	$10A$
Frobenius	0	$1A$	$5A$
Compress		$15M + 31A + I$	$290M + 1580A + I$
Decompress		$27M + 52A + I$	$470M + 2585A + I$

Figure 5.1: Arithmetic costs of operations in T_{30}.

where the preimage $x \in T_{30}(\mathbb{F}_q)$ has an order l dividing $\Phi_{30}(q)$ so we know that z has order dividing $l(q^2-q+1)$, which we can use to reduce the exponent $\alpha\Phi_{30}(q)$. As noted before, the computation of y^{-1} is virtually free, so this method is clearly preferable to the first suggestion.

5.5.3 Arithmetic Costs

Let M, A, S and I represent the cost of multiplication, addition, squaring and inversion in $\mathbb{F}_q$, respectively. In Figure 5.1 (cf. Lemma 3.2) we detail the relative costs for arithmetic in $\mathbb{F}_{q^5}$, and for both $T_6(\mathbb{F}_q)$ and $T_6(\mathbb{F}_{q^5})$. Compression and decompression are based on CEILIDH.

5.5.4 Exponentiation in T_{30}

In protocols, one is required to perform one of three operations involving exponentiation: a single exponentiation in $T_{30}(\mathbb{F}_q)$, a double exponentiation in $T_{30}(\mathbb{F}_q)$, or a single exponentiation in $T_6(\mathbb{F}_{q^5})$ (for the map $(\sigma')^{-1}$ described above). Since $T_{30}(\mathbb{F}_q) \subset T_6(\mathbb{F}_{q^5})$, we can perform all three of these in $T_6(\mathbb{F}_{q^5})$ using the methods developed in [148]; the main properties one can exploit are the degree two Frobenius automorphism and fast squaring.

In a subgroup of order l where $l \mid (q^{10} - q^5 + 1)$, in the same way as in Chapters 3 and 4, we write an exponent m as $m \equiv m_1 + m_2 q^5 \mod l$, where m_1 and m_2 are approximately half the bit-length of l, based on the method in [148], and combine these using the JSF [147]. To perform a double exponentiation, we split both exponents as with the single exponentiation, and perform the necessary four-fold multi-exponentiation as a product of two double exponentiations, combining the required squarings.

In general one may also be able to exploit the additional structure of T_{30}, which possesses an automorphism of degree eight, namely, the Frobenius automorphism. One can in principle employ exactly the same method as above and perform an eight-fold multi-exponentiation. However for the parameter sizes we consider in this chapter, we use a much simpler method based on the q-ary expansion of an exponent m. Specifically, since our value of q will be small (fitting into either one or two words), we can write an exponent m as $m = \sum m_i q^i$.

For our implementation where q^{30} has size approximately 1024 bits, exponentiating by a 160 bit exponent consists of five terms in the q-ary expansion, and hence we perform a five-fold multi-exponentiation. To perform this one can use ideas of Proos [122], which extend the JSF to more than two exponents. However due to the amount of precomputation required, for exponents of cryptographic interest we use a naive combination of the JSF and the non-adjacent form (NAF). This results in an effective exponent length of around the same size as q, significantly reducing the number of squarings needed for exponentiation.

With regard to decompression, the exponent in this case is slightly longer than for a single exponentiation. Again we use a q-ary expansion, consisting of seven terms in this instance, and apply the JSF to three pairs of them and the NAF to the remaining one.

Note that for larger parameter choices, one can clearly construct more efficient multi-exponentiation methods than those we have optimised for 1024 bit fields. We omit the details.

5.5.5 Parameter Selection

Rubin and Silverberg discuss parameter selection for T_{30} in Section 3.10 of [128]. We followed their method, starting with primes $p \equiv 1 \bmod 30$ of about 30 (resp., 61) bits, finding 32-bit (resp., 64-bit) primes q of order 30 mod p such that $q \equiv 2$ mod 9 and $q \equiv 7$ mod 11, then using the Elliptic Curve Method to remove small

q	l
2229155309	931607823866669709267930039057677132828697751771
2527138379	963373263959318090938089232997832791220899903311
2559356147	922800311037389261880873570251585702571121590451
2925130259	899122187666688780457417063691715267976198516591
3020282723	734463532846449031549478184170595775906318188901
3734718203	789572131486790156853093352977895757720566978441

Figure 5.2: Parameter examples with 32-bit primes q

prime factors from $\Phi_{30}(q)/p$, and checking to see if what remains is a prime of about 160 (resp., 200) bits. This results in parameters with q a 32-bit (resp., 64-bit) prime with the property that the order of $T_{30}(\mathbb{F}_q)$ is divisible by a 160-bit (resp., 200-bit) prime l. These choices give security equivalent to 960-bit (resp., 1920-bit) RSA security.

By suitably optimizing the Elliptic Curve Method parameter choices, we were able to generate parameters at a rate of about one example every minute or two for 32-bit primes q (with 160-bit primes l), using a Macintosh G5 dual 2.5GHz computer. For 64-bit primes q, we obtained examples with l a prime of between 198 and 202 bits at a rate of one every few hours. The parameters are like Diffie-Hellman parameters, in the sense that the same parameters can be used for all users, and for many applications do not need to be changed frequently. In Figure 5.2 we list some examples with 32-bit primes q and 160-bit primes l. In Figure 5.3 we list some examples with 64-bit primes q and 200-bit primes l.

5.6 Implementation Results

In order to understand the real-world performance of our construction, we implemented the entire system and ran a number of timing experiments. Our main goal is to compare the performance of an implementation of $T_6(\mathbb{F}_{q_L})$ using CEILIDH against an implementation of $T_{30}(\mathbb{F}_{q_S})$ of similar cardinality using our construction. Here, we denote the special cases of large and small q as q_L and q_S. We used $\log_2(q_L) \approx 5 \cdot \log_2(q_S) \approx 176$ bits, so that in both cases, there is a subgroup

q_1	9909125592335111369
q_2	10640772970658245433
q_3	11042402719715204339
q_4	11391285666382073129
q_5	11868436123416952031
q_6	17174393702711641469

l_1	1056384871088595423227115173568048528184621140903052910805301
l_2	3170119585137777422832938014760851013504258723575431018642871
l_3	1179345732085674283659621603717770735788409366766144466686061
l_4	1293678412210548537320558698939727346786705884728706067133651
l_6	1230352242796051691760643717809792393751225110105630495113071
l_6	1070675878645369998848869455205552403869773154208635001001721

Figure 5.3: Parameter examples with 64-bit primes q and corresponding l of about 200-bits.

of roughly $\log_2(l) \approx 160$ bits in size. These parameters heuristically provide the equivalent of 1024-bit RSA security (cf. Chapter 9 for an analysis of this assumption).

We constructed our implementation entirely in C++, apart from small sequences of assembly language to accelerate arithmetic in $\mathbb{F}_q$, using the GCC $3.4.2$ compiler suite. The timing experiments were carried out on a Linux based PC housing a 2.8 GHz Intel Pentium 4 processor and 1 GB of memory. We selected our system parameters as in Section 5.5.5. In all of our timing experiments we generated random operands and averaged the timings of many experiments to get a representative result. Note that exponents are reduced modulo l in all cases. Our sliding window had a maximum size of four.

Figure 5.4 shows timings for basic field and torus arithmetic. Arithmetic in $\mathbb{F}_{q_L}$ is used in $T_6(\mathbb{F}_{q_L})$ and arithmetic in $\mathbb{F}_{q_S}$ and $\mathbb{F}_{q_S^5}$ is used in $T_{30}(\mathbb{F}_{q_S})$. Figure 5.5 details the cost of mapping between different representations (compress and decompress) and the cost of different exponentiation methods which might be used within an actual cryptosystem.

It is difficult to get an exact comparison with other work on ECC and XTR, partly because of differences in host processor and levels of optimisation used

	$\mathbb{F}_{q_L}$	$\mathbb{F}_{q_S}$	$\mathbb{F}_{q_S^5}$	$T_6(\mathbb{F}_{q_L})$	$T_{30}(\mathbb{F}_{q_S})$
Addition	$0.80\mu s$	$0.52\mu s$	$0.82\mu s$		
Frobenius			$0.48\mu s$	$1.64\mu s$	$1.10\mu s$
Square	$2.51\mu s$	$0.61\mu s$		$13.80\mu s$	$21.61\mu s$
Multiply	$2.58\mu s$	$0.62\mu s$	$3.78\mu s$	$32.30\mu s$	$65.92\mu s$
Inverse	$92.71\mu s$	$2.04\mu s$	$16.03\mu s$	$1.82\mu s$	$1.29\mu s$

Figure 5.4: Timings of basic field and torus arithmetic.

	$T_6(\mathbb{F}_{q_L})$	$T_{30}(\mathbb{F}_{q_S})$
Compression		
Compress	$131.30\mu s$	$0.13ms$
Decompress	$188.61\mu s$	$4.92ms$
Exponentiation		
Binary	$5.21ms$	$9.12ms$
Sliding Window	$4.39ms$	$7.53ms$
q-ary		$3.11ms$
JSF Single	$2.79ms$	$4.57ms$

Figure 5.5: Timings for compression, decompression and exponentiations.

by different authors in producing benchmark timings. However, a comparison with the highly optimised ECC results of Avanzi [4], for example, gives some insight. For similar levels of security, direct comparison shows an exponentiation in $T_{30}(\mathbb{F}_{q_S})$ is only around twice as costly as an ECC point multiplication; correcting for the difference in processors still means that $T_{30}(\mathbb{F}_{q_S})$ is at least competitive. The case of XTR is easier to compare against since we essentially use the same experimental platform as that given in Chapter 3. It turns out that XTR is marginally faster. Solely from the point of view of performance, we conclude that our construction is a competitive alternative to existing cryptosystems.

5.7 Concluding Remarks

In this chapter we constructed an efficient "almost bijection" between $T_{30}(\mathbb{F}_q) \times \mathbb{A}^2(\mathbb{F}_q)$ and $\mathbb{A}^{10}(\mathbb{F}_q)$ which achieves better compression than XTR and CEILIDH for the compression of as few as two group elements. In particular, we obtained ElGamal ciphertexts that are 10% smaller than in previous schemes. We also developed an efficient implementation, using a variety of techniques for reducing the computational requirements and obtaining a scheme much more practical than that in [157]. From experimental results we conclude that our construction is a competitive alternative to the best existing public key cryptosystems.

Part II

Security Analysis

Chapter 6

The Discrete Logarithm Problem

In this expository chapter we provide a brief history of the study of the discrete logarithm problem (DLP), make some pertinent observations, and put into context the contributions detailed in Chapters 7, 8 and 9.

6.1 The Discrete Logarithm Problem

Given a finite cyclic group G with group operation '$\cdot$', written multiplicatively, the DLP in $(G, \cdot)$ is to compute, for a given element $h \in G$ and generator $g \in G$, the unique integer $x \bmod \#G - 1$ such that $g^x = h$. In analogy with logarithms in $\mathbb{R}$, we represent this equality also as $\log_g h = x$.

In the case that the group G is the multiplicative group of a finite field, which is the main subject of Part II of this thesis, much can be said about the problem. In Section 6.4 we give a brief synopsis of the subject's rich history in this setting, leaving the details of some contemporary methods to the following chapters, while in this chapter we make some elementary statements regarding the DLP in an arbitrary finite cyclic group.

One issue of primary importance for the DLP is that of group representation. While all cyclic groups of order n are isomorphic (to $(\mathbb{Z}/n\mathbb{Z}, +)$), the particular representation of elements in a given instantiation of the problem may obscure this cyclic structure. Therefore it is the difficulty of computing this isomorphism that makes some DLPs hard, such as in suitable elliptic curve groups, and some easy, such as in $(\mathbb{Z}/n\mathbb{Z}, +)$ itself.

The significance of this observation, which may not initially appear to be all that deep, is motivated by a natural partitioning of the set of algorithms to solve the DLP into two distinct classes. Firstly, we have the so-called *generic algorithms*, which apply to any finite cyclic group and exploit only the existence of the group operation, and are blind to the group representation. These are of exponential complexity, typically with exponent one half, and hence are also known as 'square-root algorithms'. The second class of algorithms, the *index calculus algorithms*, make essential use of the group representation, and also some notion of factorisation. These algorithms, having subexponential complexity, are generally far more efficient than their generic counterparts in cryptographically relevant cases.

Before proceeding, we should specify the notion of subexponentiality we assume. Essentially, the following function allows interpolations between polynomial, and exponential complexity:

$$L_q[\alpha, c] = \exp((c + o(1)) \log(q)^{\alpha} \log(\log(q))^{1-\alpha}).$$

The value $\alpha \in [0, 1]$ gives a measure of subexponentiality, with $\alpha = 0$ being polynomial time in the size of q (and c the degree of this polynomial), and $\alpha = 1$ corresponding to fully exponential run-time. Since α is the important variable, sometimes we will drop the c from complexity statements.

A result of particular importance in this area is one due to Shoup [142] and Nechaev [107], which asserts that within the *generic group model*, the best discrete logarithm algorithms one can obtain are necessarily of full exponential complexity, with exponent one half. This model is a formalisation of the notion alluded to above, stipulating that group elements are arbitrarily, rather than naturally encoded, and hence one can utilise no information regarding element representation. Shoup's result therefore assures one that in order to obtain a subexponential algorithm, one *must* exploit the given group representation. As a good rule of thumb, the more 'structure' there is available in a particular group representation, the more likely it is that there exists a corresponding algorithm that can exploit this structure to solve DLPs more efficiently.

In the next section, we give a short description of a simple reduction of the DLP, and provide suitable references to the generic, or square-root algorithms, since these are not the focus of this chapter. Then in Section 6.3 we explain in detail the principle of the index calculus method, and in Section 6.4, outline specific algorithms for solving the DLP in finite fields.

6.2 General Methods

Besides the general methods available to solve the DLP, if the factorisation of the group order is known, one can apply the following simple observation of Pohlig and Hellman [119] to reduce the problem to one of prime order subgroups.

6.2.1 The Pohlig-Hellman Simplifi cation

Let $\#G = n$ and suppose the prime factorisation of n is known. The idea is to solve the DLP modulo each prime power dividing n, and then recover the full discrete logarithm via the Chinese Remainder Theorem (CRT).

Suppose we want to solve $g^x = h$ for x. For $p \mid n$, we project the DLP into the subgroup of order p by powering g and h by $n' = n/p^{e-1}$, giving g' and h' say, where p^e is the largest power of p which divides n. We then solve

$$g'^{x_0} = h',$$

for $x_0 \equiv x \bmod p$. In order to find $x \bmod p^e$ we proceed as follows. Suppose we know $x \equiv x_i \bmod p^i$. Then $x = x_i + \lambda p^i$ for some $\lambda \in \mathbb{Z}$. Consequently,

$$r = (h \cdot g^{-x_i}) = (g^{p^i})^{\lambda} = s^{\lambda},$$

where s has order n/p^i. Then powering r and s by n/p^{c-i-1}, one can compute $\lambda \bmod p$ by employing the same method as before. Therefore each coefficient of the base p expansion of $x \bmod p^e$ can be computed by solving a DLP for an element of order p only. Combining the modular information with the CRT then solves the original DLP.

If $n = \prod p_i^{e_i}$ and we use one of the square-root algorithms described in the next subsection, the complexity of the method is $O(\sum e_i(\log n + \sqrt{p_i}))$ group operations. One can deduce from this that for cryptographic purposes, one necessary requirement of a secure group is that its order is divisible by a prime of a sufficient size to ensure the desired level of security. So for instance, groups whose orders are divisible by small primes only, are completely insecure.

6.2.2 Square-root Algorithms

The two square root algorithms we mention here are general purpose, in that they can be applied to arbitrary finite cyclic groups. From Shoup's result [142], one might conclude that for a DLP based in any real instantiation of a group, there should exist a more efficient algorithm to solve the DLP. Surprisingly, for the DLP on elliptic curves defined over prime fields, square root algorithms are the best known (except for a few easily identified cases - see [10]), simply because no one has yet thought of a sufficiently intelligent method to exploit the group representation employed.

Broadly speaking, there are essentially only two flavours to the square root algorithms. Firstly, we have a deterministic method, known as the *baby step/giant step* (BS/GS) method, which is due to Shanks in the context of integer factorisation algorithms and class group computations [141]. This utilises a standard space/time trade-off, and has deterministic complexity $O(\sqrt{n})$ for a group of order n. The drawback of this method is that it requires $O(\sqrt{n})$ memory, which imposes serious hardware demands. We refer the reader to chapter 5 of [10] for a description of both the BG/GS method, as well as the following method.

In order to overcome this constraint, in 1978 Pollard [120] suggested a probabilistic method, based on the Birthday Paradox. Also known as Pollard's Rho Method, because the random walks employed to find a collision may be visualised as tracing out the Greek letter ρ, the expected time for the method to compute discrete logarithms is $O(\sqrt{n})$ group operations also, however the algorithm has virtually no memory overhead. Hence with any hardware constraints the method is preferable to BS/GS. While we choose not to go into the details here, we note that like BS/GS, the method can be efficiently parallelised [151].

Since parallelised Pollard rho is the best publically known method to solve the ECDLP over prime fields (and binary fields with prime extension degree), much research has focused on developing optimised implementations. Indeed, several large scale experiments have been conducted [152], lending assurance of the expected hardness of solving the ECDLP, at least when using this method.

6.3 The Index Calculus Method

Currently the most effective principle for computing discrete logarithms for non-trivial group representations is the Index Calculus Method (ICM). The term heralds from the 18th century when the discrete logarithm of an integer modulo p relative to a primitive root was also known as its index, and it is this index which is being calculated.

The use of the definite article is perhaps a misnomer since there are many ways to compute logarithms, however the principle underlying the method which is now synonymous with the name is currently the best known and applies in a broad range of circumstances, and so in this sense the term is justified, if a little overstated.

Crucially, the method relies on the ability to factorise into some notion of 'primes', elements in a particular representation of the group in which logarithms are to be computed. One also requires a notion of size for elements, or a norm function $N : G \mapsto \mathbb{N}$, and typically a suitable norm can be defined as soon as one has a suitable notion of primes.

Using these two concepts, for an integer B, one defines an element to be 'B-smooth' if it factors into a product of primes, all of whose norms are less than or equal to B. It is the existence of a 'sufficient proportion' of smooth elements in a group which makes the ICM work, as follows.

At a high level, the ICM consists of three phases. Fix a group G of order n as before. For the first phase one first fixes a subset $\mathcal{F}$ of G called the factor base,

$$\mathcal{F} = \{p_1, \ldots, p_{|\mathcal{F}|}\},$$

consisting of all prime elements with norm $\leq B$ for some bound B. The reason the factor base is chosen as such is that the number of elements of G generated by this set is larger than that generated by any other set of the same cardinality. Hence, with this choice, heuristically the probability that a random element of G factors over $\mathcal{F}$ is maximised. One then generates multiplicative relations between elements of G in the hope that they factor completely over $\mathcal{F}$. This can be performed simply by computing random powers of a generator g^k and checking these for smoothness or in some situations, such as for finite fields, more sophisticated methods can be brought to bare.

The purpose of the second phase is then to compute the discrete logarithms of all the factor base elements, as follows. Upon taking logarithms with respect to a generator g one obtains a set of equations of the form

$$\sum_{(j_i,p_i)\in\mathbb{Z}\times\mathcal{F}} j_i \log_g p_i \equiv 0 \mod n. \tag{6.1}$$

Presuming the resulting linear system is of full rank, i.e., sufficiently many relations have been collected during the first phase, one inverts this system modulo n to obtain the quantities $\log_g p_i$ for each $p_i \in \mathcal{F}$. If the prime factors of n are known this computation can be simplified by solving for the p_i modulo each prime divisor of n, and then reconstructing their values modulo n using the CRT.

The final phase then consists of computing discrete logarithms for arbitrary elements of G, and once the first two phases are complete, can be usually be performed relatively quickly. For $h \in G \setminus \mathcal{F}$, one computes $g^l h$ for random integers l until one factors over $\mathcal{F}$, yielding the solution

$$\log_g h = \Big\{ -l + \sum_{(k_i,p_i)\in\mathbb{Z}\times\mathcal{F}} k_i \log_g p_i \Big\} \mod n.$$

This is the core technique underlying all index calculus methods, although the details for a given group representation involve various notions of factorisation, and correspondingly different norms. From an efficiency perspective, there are three main issues to address. By what method should one collect relations? What is the most time-efficient size for the factor base? Can one do better than randomly guessing in the third phase? For different groups these issues naturally lead to different considerations and techniques.

The central idea of using a factor base and exploiting smoothness is straightforward and consequently, has been independently discovered many times (see [87, 98, 113, 114] and the references therein). In the context of the DLP in prime fields, it was first published by Kraitchik in the 1920's [78, 79].

As an example of the applicability of the ICM, Enge and Gaudry [34] have unified its various instantiations in several contexts, using a sufficently general description of the problems. Within this framework, they proved an expected $L[1/2]$ running time for several algorithms simultaneously. While for finite fields, the Number Field Sieve and Function Field Sieve algorithms have heuristic $L[1/3]$ expected running time, the ICM principle is still used, and barring a dramatic development in DLP algorithms, appears to be the best currently available method.

6.4 The Discrete Logarithm Problem in Finite Fields

The most successful algorithms for solving the discrete logarithm problem in a finite field $\mathbb{F}_{p^m}$ are the Number Field Sieve (NFS) [52, 86, 132], and the Function Field Sieve (FFS) [1,3], which are applicable for $m < (\log p)^{1/2}$ and $m > (\log p)^2$ respectively[1]. Both have conjectured complexity $L[1/3, c]$, where c ranges between $(32/9)^{1/3}$ and $(64/9)^{1/3}$ for the NFS [130], and is $(32/9)^{1/3}$ for the FFS [70].

The reason the subexponentiality constant is smaller than one half, as seems to be the bound for all other discrete logarithm problems, is essentially due to the fact that in a field, there exists a second binary operation between elements, namely addition, which can be exploited.

[1]See Section 9.6 for comments regarding recent advances in this area.

One naive way to exploit this second operation is to express the discrete logarithm function as a polynomial [99] (which is always possible for a function from a field to itself):

$$\log_g h = \sum_{m=0}^{n-1} \left\{ \sum_{i=1}^{q-2} h^i (1 - g^i)^{-p^m} \right\} p^m,$$

although this is clearly less effective than a naive enumeration of powers of g, as promising as it may look at first sight. Fortunately, more sophisticated methods are available.

In order to beat the '$L[1/2]$ barrier', one needs a better way to generate relations than simply choosing random elements and trying to factor them over the factor base. In 1984 Coppersmith gave the first heuristic $L[1/3]$ algorithm in the case of characterstic two fields [24, 25, 113]. For the DLP over large prime fields, Gordon observed in 1992 that the well-known NFS could be adpated to compute logarithms in $\mathbb{F}_p$ [52], obtaining a complexity of $L[1/3]$ also. Then in 1994, Adleman designed the FFS in analogy with the NFS [1]. Later work by Schirokauer [132] then generalised the NFS method to compute logarithms in finite fields with very small extension degree. These last two results are responsible for the aforementioned constraints on the base field size, and extension degree, p and m. It is still an open problem to find an algorithm with heuristic $L[1/3]$ complexity for *all* finite fields. We note that Nguyen has given an interpretation of both the NFS and FFS in the context of computing discrete logarithms in finite fields, in terms of the arithmetic of relative Brauer groups [109], which may lead to developments in answers to this problem.

This part of the thesis is dedicated to the security aspects of the systems detailed or proposed in Part I. In Chapters 7 and 8 we give a detailed description of the FFS, a probablistic model of its behaviour, and a corresponding implementation in characteristic three. In Chapter 9 we propose a completely new approach to solving the discrete logarithm on algebraic tori (and hence in extension fields), which is independent from the described field-based methods, and analyse its performance.

Chapter 7

Estimates for Discrete Logarithm Computations in Finite Fields of Small Characteristic

Motivated by the use of characteristic two and three fields in identity-based cryptography, using an elementary probabilistic model of a version of the Function Field Sieve (FFS), in this chapter we give estimates for the running-time of discrete logarithm computations in $\mathbb{F}_{p^n}^{\times}$, for small p.

The results of this chapter appeared in [54].

7.1 Introduction

The proposed use of elliptic curves with low embedding degree for identity-based cryptography has caused some consternation within the cryptographic community, primarily due to the uncertain security assurances they provide. Indeed, when Menezes, Okamoto and Vanstone [100] proved in 1993 that the ECDLP is no harder than the DLP in some extension field, one immediate unaminous presumption was that supersingular elliptic curves, which have embedding degree ≤ 6, are weak, and hence are unsuitable for cryptographic purposes.

However, the proposal of the first efficient identity-based key agreement, signature and encryption schemes [14,129] relied upon the existence of elliptic curves with low embedding degree. Therefore cryptographers were forced to re-examine the presumption of insecurity previously made about such curves, if they were to benefit from the radically new functionality afforded by pairings (see [153] for an impressive collection of pairing-related references).

For contemporary security parameters, there are two types of elliptic curves to consider. The use of supersingular elliptic curves in identity-based cryptography has been proposed since they can provide some efficiency and implemention advantages [6, 7, 30, 43] over, and differing properties from, the so-called MNT curves [104], which form a family of elliptic curves over prime fields with embedding degree six. Indeed, a comparison between characteristic three pairings and pairings on MNT curves suggests that each have advantages for different applications [118]. For prime fields of characteristic $p > 3$, supersingular curves have maximum embedding degree two, while for characteristics two and three, the maximums are four and six, respectively [43]. The latter two are therefore preferable from an efficiency perspective since one can then keep the field of definition of the elliptic curve relatively small.

Despite these potential advantages, the central drawback of using small characteristic fields is that in anology with the relationship between the special, and general NFS [86], discrete logarithms can be computed much faster in these fields than for prime fields of a similar size [24,25,131]. Thus in order to attain the same level of security, one must increase the size of the field of definition of the curve. By how much one should do so has yet to be fully analysed.

It is the goal of this chapter to help provide a method to establish the level of security provided by a given field. While we do not conclusively assess the security of various small characteristic fields relative to the best known algorithm - the Function Field Sieve (FFS), which has complexity $L[1/3, (32/9)^{1/3}]$ - we do describe a model of its behaviour which can be specialised for a particular implementation, and adjusted accordingly.

In any large-scale implementation of the FFS it is essential to solve many small example DLPs to gain an idea of what parameter choices are most effective. This initial understanding of the performance of an implementation can then be used to extrapolate optimal parameter choices to the larger, more costly example. To date there are only very few published accounts of discrete logarithm computations undertaken in reasonably large fields in characteristic two [53, 70, 155], and none in fields of other small characteristics. Considering that the current record for the field $\mathbb{F}_{2^{607}}$ [155] was the result of over a year's worth of highly parallelised computation, rather than implementing another example, we develop a detailed model of the behaviour of the FFS that allows one to make practical estimates of optimal parameters and expected running-times for any $\mathbb{F}_q$ where the FFS applies.

In particular, we obtain sharp probability estimates that allow us to select optimal parameters in cases of cryptographic interest, without appealing to the heuristics commonly relied upon in an asymptotic analysis. We also consider the possible effect of different field representations, when the extension degree is composite, and indicate that in some cases these may offer an advantage to the attacker.

Currently the linear algebra step is considered to be a bottleneck in the index calculus method, due to the difficulty of parallelising this stage efficiently, although some progress has been made [116, 162]. Although the situation is improving, the heuristically optimum factor base sizes are still beyond what can adequately be accommodated in the matrix step. One question we answer is how varying the size of the factor base, which is the main constraint in the linear algebra step, affects the efficiency of the FFS. It is the notion of 'smoothness' that allows us to assess the probability that a random element will factor in the factor base, and this is based solely on the degree of that element. Consequently if the factor base is chosen to consist of all monic irreducible polynomials up to a certain degree bound, then although we can make good estimates of smoothness probabilities, we are restricted in the choice of the size of the factor base. In higher characteristics this problem renders typical estimates of smoothness probabilities useless.

Ideally for an optimisation analysis we would like to allow the size of the factor base to vary over $\mathbb{N}$, but it is essential that we have highly accurate estimates for the probabilities involved. Such an interpolation is often used in practice, yet no formal analysis has been given. We give such an analysis, allowing implementors to decide whether such an interpolation is useful in practice.

We also give evidence that for any fixed field size some may be weaker than others of a different characteristic or field representation, and compare the relative difficulty of computing discrete logarithms via the FFS in such cases.

The model we assume in our running-time estimates is the version of the FFS as detailed in [3]. In order to assess these times accurately, it is necessary to provide in detail an analysis of how we expect a basic implementation of the FFS to behave in practice. In the next section we briefly explain the general principle of the FFS. In Section 7.3 we give the revised probability estimates we need and describe our model. In Section 7.4 we present our results, and in Section 7.5, we draw some conclusions.

7.2 The Function Field Sieve

The idea of using a function field to obtain relations of the form (6.1) is originally due to Adleman [1] and generalises Coppersmith's early algorithm [24,25] for the characteristic two case. Here we give a brief overview of the algorithm. Note that in Chapter 8 we give a more detailed description of the FFS method.

Letting $q = p^n$, we choose as our representation of $\mathbb{F}_q$ the set of equivalence classes in the quotient of the ring $\mathbb{F}_p[t]$ by one of its maximal ideals $f(t)\mathbb{F}_p[t]$, where $f(t)$ is an irreducible polynomial of degree n. Each equivalence class can then be uniquely identified with a polynomial in $\mathbb{F}_p[t]$ of degree strictly less than n. Arithmetic is performed as in $\mathbb{F}_p[t]$, except that when we multiply two elements we reduce the result modulo $f(t)$.

To begin with one selects a polynomial μ in $\mathbb{F}_p[t]$, and an absolutely irreducible bivariate polynomial $H(t, X)$ such that $H(t, \mu(t)) = 0 \mod f(t)$, where $f(t)$ is the irreducible polynomial in $\mathbb{F}_p[t]$ which defines $\mathbb{F}_q$. This condition provides a map ϕ from the ring $\mathbb{F}_p[t, X]/(H)$ to $\mathbb{F}_q^{\times}$ induced by sending $X \mapsto \mu$. Given $H(t, X)$, its function field L is defined as the field of fractions of $(\mathbb{F}_p[t, X]/(H))$.

In the FFS we have two factor bases. The small degree monic irreducible polynomials in $\mathbb{F}_p[t]$ form the rational factor base $\mathcal{F}_R$, while those places in L that lie above the primes of $\mathcal{F}_R$ and are of degree $\leq$ the maximum degree of elements of $\mathcal{F}_R$, form the algebraic factor base $\mathcal{F}_A$. For a given coprime pair $(r, s) \in \mathbb{F}_p[t]^2$, we check if $r\mu + s$ decomposes on the rational side as a product of primes from $\mathcal{F}_R$, and also whether the divisor associated to the function $rX + s$ decomposes as a sum of places in $\mathcal{F}_A$ on the algebraic side. To verify the second condition we need only compute the norm of $rX + s$ over $\mathbb{F}_p[t]$, $r^d H(t, -s/r)$, where d is the degree in X of H, and check this for smoothness in $\mathcal{F}_R$.

Provided that $H(t, X)$ satisfies eight technical conditions given by Adleman, we obtain a relation in $\mathbb{F}_q^\times$ by raising the function $rX+s$ to the power h_L, the class number of L, and applying the morphism ϕ to the components of its associated divisor, and also to $rX + s$, giving $r\mu + s$. One point to note is that for each element of the algebraic factor base $\mathcal{F}_A$ occuring in the decomposition of the divisor associated to $(rX + s)^{h_L}$, applying the morphism ϕ gives an additional logarithm for an element of $\mathbb{F}_q^\times$ which is unlikely to be in the rational factor base (and which we need not compute explicitly). Therefore, with regard to the linear algebra step of the algorithm, we expect the matrix to have about $\mathcal{F}_R + \mathcal{F}_A$ rows, and once the elimination has been performed, we may disregard the logarithms of these superfluous elements.

The sieving stage of the FFS refers to various methods which may be employed to enhance the probability that the pairs (r, s) will be coprime and 'doubly smooth', see [24, 46, 113]. For simplicity we do not incorporate these methods into our probabilistic model, although one does expect them to be a significant factor in actual run times. With regard to estimating suitable parameter choices for discrete logarithm computations in various fields, the results of Chapter 8 imply that this simplification still preserves the accuracy of these estimates, if not their complete precision.

In Section 7.3.2, we describe the properties of the curve $H(t, X)$ that are essential to our model. Note also that we ignore phase three of the index calculus method, since in practice the time for computing individual logarithms is negligible [1, 24, 113, 131],

7.3 Methodology of our Analysis

In this section we describe the methodology of our calculations. Our goal here is twofold: primarily we want to ensure that our model portrays as accurately as possible the behaviour of a practical implementation of the FFS; and secondly to ensure that the full spectrum of choice for the relevant variables is thoroughly investigated. With regard to the former, we first give a refinement of the well-known smoothness probability function [113].

7.3.1 Some Factorisation Probabilities

Let $\eta = |\mathcal{F}_R|$ and let

$$I_p(j) = \frac{1}{j} \sum_{d|j} \mu(d) p^{j/d} \tag{7.1}$$

be the number of monic irreducible polynomials in $\mathbb{F}_p[t]$ of degree j, where μ is the mobius function. This then determines a unique $m \in \mathbb{N}$ such that

$$\sum_{j=1}^{m} I_p(j) \leq \eta < \sum_{j=1}^{m+1} I_p(j)$$

so that while we have all irreducible polynomials in $\mathbb{F}_p[t]$ of degrees $\leq m$ in our factor base, we are also free to select a fraction α of primes of degree $m+1$, with

$$\alpha = (\eta - \sum_{j=1}^{m} I_p(j))/I_p(m+1).$$

Such a proportion α is used in some implementations of the FFS, and in the following we investigate the practical implications of such a parameter. We assume that the additional degree $m+1$ members of the factor base have been chosen at random, before sieving begins. In an implementation these would probably be selected dynamically, as the sieve progressed, though we do not believe this would affect these estimates significantly.

Definition 7.1. *Let $\rho_{p,\alpha}(k,m)$ be the probability that a given monic polynomial in $\mathbb{F}_p[t]$ of degree k has all of its irreducible factors of degrees $\leq m+1$, and that those of degree $m+1$ are contained in a proportion α of preselected irreducible polynomials of degree $m+1$.*

We implicitly assume throughout, as do all authors on this subject, that elements of $\mathbb{F}_p[t]$ behave like independent random variables with respect to the property of being smooth. Provided we have a process to generate elements uniformly, this is reasonable. Note that the case $p = 2, \alpha = 0$ is the definition of $\rho(k,m)$ in [113], which can be computed using the counting function we introduce in Definition 7.2, and with which we derive an exact expression for $\rho_{p,\alpha}(k,m)$. When a given polynomial factors within this extended factor base we say it is (m,α)-smooth.

Definition 7.2. *Let $N_p(k,m)$ be the number of monic polynomials $e(t) \in \mathbb{F}_p[t]$ of degree k such that $e(t)$ has all of its irreducible factors of degrees $\leq m$, i.e.,*

$$e(t) = \prod_i e_i(t)^{\beta_i},\ deg(e_i(t)) \leq m.$$

For exactness we further define $N_p(k,0) = 0$ for $k > 0$, $N_p(k,m) = p^k$ if $k \leq m$, and $N_p(k,m) = 0$ if $k < 0$ and $m \geq 0$.

We are now ready to state

Theorem 7.1.

i. $$N_p(k,m) = \sum_{n=1}^{m} \sum_{r \geq 1} N_p(k-nr, n-1) \binom{r + I_p(n) - 1}{r},$$

ii. $$\rho_{p,\alpha}(k,m) = \sum_{r \geq 0} \frac{N_p(k - r(m+1), m)}{p^k} \binom{r + I_p(m+1) - 1}{r} \alpha^r.$$

Proof. The proof of (i) is virtually identical to the derivation of $\rho_{2,0}(k,m)$ in [113], since in $\mathbb{F}_2[t]$ all non-zero polynomials are monic. For (ii) let $e(t)$ be any monic polynomial in $\mathbb{F}_p[t]$ of degree k, all of whose irreducible factors are of degrees $\leq m+1$. Such a polynomial can be written uniquely as

$$e(t) = g(t) \prod_{u(t)} u(t)^{\beta_{u(t)}},$$

where the $u(t)$ are monic and of degree $m+1$, $\sum \beta_{u(t)} = r$ for some $r \in \mathbb{N}$, and $g(t)$ is a monic polynomial of degree $k - r(m+1)$, all of whose prime factors are of degrees $\leq m$. Given $m+1$ and r, there are $N_p(k-r(m+1), m)$ such $g(t)$, and the number of such $\prod u(t)^{\beta_{u(t)}}$ is the number of $I_p(m+1)$-tuples of non-negative integers which sum to r (since we have $I_p(m+1)$ possibilities for the $u(t)$), which is just

$$\binom{r + I_p(m+1) - 1}{r}.$$

So for a given r, the probability that a monic polynomial $e(t) \in \mathbb{F}_p[t]$ of degree exactly k has all its irreducible factors of degrees $\leq m+1$, exactly r of its irreducible factors having degree $m+1$, and that these are in the preselected factor base is then

$$\frac{N_p(k - r(m+1), m)}{p^k} \binom{r + I_p(m+1) - 1}{r} \alpha^r,$$

since there are p^k monic polynomials of degree k. Hence the total probability that $e(t)$ has all its irreducible factors in the chosen factor base is

$$\sum_{r \geq 0} \frac{N_p(k - r(m+1), m)}{p^k} \binom{r + I_p(m+1) - 1}{r} \alpha^r,$$

which is our theorem.

□

Remark. We have noted already that for $\alpha = 0$, we obtain $\rho_p(k,m)$ (assuming $0^0 = 1$; while for $\alpha = 1$, we obtain

$$\rho_p(k,m) + \sum_{r \geq 1} \frac{N_p(k - r(m+1), m)}{p^k} \binom{r + I_p(m+1) - 1}{r},$$

which, by the recurrence (i) is equal to

$$\rho_p(k,m) + \frac{1}{p^k}\left\{N_p(k,m+1) - N_p(k,m)\right\} = \rho_p(k,m+1),$$

verifying our calculation.

We will also need the following simple theorem in the next section.

Theorem 7.2. *Let $a_{R,S}$ be the number of coprime pairs of polynomials (r,s) of degrees $0 \le R \le S$ with r monic. Then*

$$a_{R,S} = \begin{cases} (p-1)^2 p^{R+S-1} & R,S > 0 \\ (p-1)p^S & \textit{otherwise}. \end{cases} \tag{7.2}$$

Proof. Considering first just monic polynomials, let $0 \le R \le S$. Since there are p^R monic polynomials of degree R, and p^S monic polynomials of degree S, there are p^{R+S} pairs of monic polynomials in this range. Let $\hat{a}_{R,S}$ be the number of monic polynomial pairs (r,s) with degrees R and S such that $\gcd(r,s) = 1$, and for each pair (r,s), let $h = \gcd(r,s)$ where $0 \le k = \delta(h) \le R$. There are p^k possible such monic h. Furthermore since $\gcd(r/h, s/h) = 1$, there are $\hat{a}_{R-k,S-k}$ possibilities for the pair$(r/h, s/h)$. Summing these possibilities over k we obtain the recurrence relation

$$p^{R+S} = \sum_{k=0}^{R} \hat{a}_{R-k,S-k} p^k.$$

Noting that $\hat{a}_{0,S-R} = p^{S-R}$ we see this has the solution

$$\hat{a}_{R,S} = \begin{cases} (p-1)p^{R+S-1} & 0 < R \le S \\ p^S & R = 0 \end{cases}$$

If we allow s to be non-monic then we simply multiply $\hat{a}_{R,S}$ by $|\mathbb{F}_p^{\times}|$, giving the stated result.

□

7.3.2 Model of the Function Field Sieve

In this section we describe the details of the model we use in our estimates, which is based on the FFS as set forth in [3].

Let d be the degree in X of the curve $H(t,X)$, and let $d' = \lceil n/d \rceil$, where n is the extension degree of $\mathbb{F}_q$ over $\mathbb{F}_p$. Let $\delta(\cdot)$ be the degree function. The irreducible polynomial f of degree n which defines $\mathbb{F}_q$ is chosen to be of the form $f(t) = t^n + \hat{f}(t)$, where $\delta(\hat{f}) \approx \log_p n$. We make this last assertion since by (7.1) the proportion of degree n monic polynomials in $\mathbb{F}_p[t]$ that are irreducible is about $1/n$, and so we expect to be able to find one satisfying $\delta(\hat{f}) \leq \log_p n$. A further condition on $\hat{f}$ is that it has at least one root of multiplicity one, so that the curve $H(t,X) = X^d + t^{dd'-n}\hat{f}(t)$ is absolutely irreducible. We let $\mu(t) = t^{d'}$ and it is easily checked then that $H(t,\mu(t)) \equiv 0 \bmod f$. Lastly we assume that we can easily find such curves for each set of parameter values we analyse, that furthermore satisfy the remaining technical requirements.

Within the model itself, we are slightly conservative in some approximations. This means our final estimates for the running-times will tend to err on the upper side of the average case, so that we know with a fair degree of certainty that given a final estimate, we can compute discrete logarithms within this time. We take as the unit run-time of our calculations the time to perform a basic field operation in $\mathbb{F}_p$, but do not discriminate between which ones as this will only introduce a logarithmic factor into the estimates. Furthermore, for a real implementation the computational difference for different p will often be irrelevant, since all small p will fit in the word size, and for characteristics two and three, there will be only a small constant difference in performance.

Given a factor base size η as in Section 7.3.1, we determine the corresponding m, and α. We assume that $\mathcal{F}_A$ has the same cardinality as the set of primes we use in the factor base $\mathcal{F}_R$ on the rational side. So the set of useful logarithms that we obtain is only half the size of the matrix that we can theoretically handle in the linear algebra step.

We now consider the number of relations we expect to obtain. Since we wish to minimise the degree of the polynomials we generate, we consider those pairs

$(r, s) \in \mathbb{F}_p[t]^2$ where $\delta(r) = R, \delta(s) = S$ are as small as possible, and $R \leq S$. We refer to this collection of elements as the sieve base, and it is typically chosen to consist of all relatively prime pairs of polynomials with degrees bounded by l, a parameter to be selected. We observe though that since we are looking to generate pairs (r, s) such that $r\mu + s$ and $rX + s$ give useful relations, we may assume that either r or s is a monic polynomial, as otherwise we will obtain $p - 1$ copies of each useful pair $r\mu + s$ and $rX + s$.

For a given $\mu \in \mathbb{F}_q^\times$ of degree $\lceil n/d \rceil$ and a given pair (r, s) with degrees (R, S), $0 \leq R, S \leq l$, the degrees of the rational and algebraic sides are respectively bounded by

$$\begin{aligned} \delta_{RAT_d}(R, S) &\leq \max\{R + \lceil n/d \rceil, S\}, \\ \delta_{ALG_d}(R, S) &\leq \max\{dR + d + \log_p n, dS\}. \end{aligned}$$

As discussed above we assume both these bounds are equalities.

The probability that each of these is (m, α)-smooth, assuming they are independent, is

$$\rho_{p,\alpha}(\delta_{RAT_d}(R, S), m).\rho_{p,\alpha}(\delta_{ALG_d}(R, S), m),$$

and the number of suitable pairs (r, s) is $a_{R,S}$, given in (7.2). Since $0 \leq R \leq S \leq l$, the number of (m, α)-smooth relations we expect to obtain is

$$M(l) = \sum_{S=0}^{l} \sum_{R=0}^{S} \rho_{p,\alpha}(\delta_{RAT_d}(R, S), m).\rho_{p,\alpha}(\delta_{ALG_d}(R, S), m) a_{R,S}$$

For each such set of relations we assume that we obtain the same number of linearly independent equations amongst the corresponding logarithms. To obtain a matrix of full rank we require that this number exceeds $|\mathcal{F}_R| + |\mathcal{F}_A|$. A basic constraint therefore is

$$M(l) \geq 2\eta. \tag{7.3}$$

When this is the case, we calculate the expected running-time as follows. We estimate the running-time of a gcd calculation for a pair r, s simply as RS, as we do not presume any fast multiplication algorithm. The time for this part of the computation is therefore

$$\sum_{S=1}^{l}\sum_{R=1}^{S} RSp^{R+S}, \tag{7.4}$$

as for each (R, S) there are p^{R+S} such pairs of monic polynomials (r, s). Furthermore, with regard to factoring both the rational and algebraic sides, it only makes sense to factor both if the first one factored possesses factors lying entirely within the factor base, which cuts down the number of factorisations to be performed considerably. With this in mind and noting the form of δ_{ALG_d} and δ_{RAT_d}, we choose the rational side to be the first factored. Again we suppose a naive polynomial factoring algorithm and simply estimate the running time as δ^3. For this part of the computation the running-time is therefore

$$\sum_{S=0}^{l}\sum_{R=0}^{S} a_{R,S}\left\{\delta_{RAT_d}(R,S)^3 + \rho_{p,\alpha}(\delta_{RAT_d}(R,S), m)\delta_{ALG_d}(R,S)^3\right\}. \tag{7.5}$$

This is essentially all that we need. The process of minimising the expected running-time for each factor base size is to compute, for various pair values (d, l), the value of (7.4) and (7.5) provided the constraint (7.3) is satisfied, and simply choose the pair which gives the smallest result.

In performing a few prelimary parameter estimates with this model, it was apparent that the estimated running-times altered considerably when the optimal value for l was changed by just one. A little reflection will reveal that the process of varying l, the bound of the degrees of the sieve base elements, is analogous to the way in which we initially regarded m, the bound of the degrees of the factor base elements. To remedy this problem we interpolate between successive values of l. Let

$$L_l(i) = \lfloor p^{l+1} i/100 \rfloor,\ i = 0, ..., 100,$$

and suppose that we have $L_l(i)$ additional monic sieve base polynomials of degree $l + 1$. The choice of 100 here is arbitrary, but we did not want to extend running-times excessively, and we can view this number simply as the percentage of monic polynomials of degree $l+1$ in the sieve base, since there are p^{l+1} monic polynomials of degree $l+1$. This gives us an additional number of expected doubly-smooth relations $M_+(L_l(i))$ equal to

$$\sum_{R=0}^{l+1} \rho_{p,\alpha}(\delta_{RAT_d}(R,l+1),m).\rho_{p,\alpha}(\delta_{ALG_d}(R,l+1),m)a_{R,l+1}L_l(i)/p^{l+1},$$

where here we have implicitly assumed that the proportion of pairs that are coprime is propagated uniformly for each interpolation value. Our constraint (7.3) then becomes

$$M(l) + M_+(L_i(l)) \geq 2\eta,$$

and the total expected running-time is then

$$\begin{aligned} T(m,\alpha,l,d,i) = \sum_{S=0}^{l}\sum_{R=0}^{S} \Big\{ & RSp^{R+S} + \delta_{RAT_d}(R,S)^3 a_{R,S} + \\ & + \rho_{p,\alpha}(\delta_{RAT_d}(R,S),m)\delta_{ALG_d}(R,S)^3 a_{R,S} \Big\} \\ + \sum_{R=0}^{l+1} \Big\{ & R(l+1)p^R L_l(i) + \delta_{RAT_d}(R,l+1)^3 a_{R,l+1} L_l(i)/p^{l+1} + \\ & + \rho_{p,\alpha}(\delta_{RAT_d}(R,l+1),m)\delta_{ALG_d}(R,l+1)^3 a_{R,l+1} L_l(i)/p^{l+1} \Big\}. \end{aligned} \tag{7.6}$$

This equation accounts for the effect of the differing degrees in sieve base pairs, the necessary gcd and factorisation computations, and for the l value interpolations. Note that this differs considerably from the

$$2\eta/\rho_{p,0}(dl + d + l + \lceil n/d \rceil + \log_p n, m)$$

typically assumed in an asymptotic analysis.

7.4 Empirical Results

In the following computations we looked at fields of characteristics 2,3 and 107, with bitsizes of 336, 485, 634, and 802. We let the size of the factor base vary from 100,000 to 2,000,000, with increments of 50,000. The correspondence was as follows:

$$\begin{array}{ccccc} \mathbb{F}_{2^{336}} & \sim & \mathbb{F}_{3^{212}} & \sim & \mathbb{F}_{107^{50}} \\ \mathbb{F}_{2^{485}} & \sim & \mathbb{F}_{3^{306}} & \sim & \mathbb{F}_{107^{72}} \\ \mathbb{F}_{2^{634}} & \sim & \mathbb{F}_{3^{400}} & \sim & \mathbb{F}_{107^{94}} \\ \mathbb{F}_{2^{802}} & \sim & \mathbb{F}_{3^{506}} & \sim & \mathbb{F}_{107^{119}} \end{array}$$

These bit-lengths were chosen so that for each pair of characteristics p, r and for each corresponding pair of extension degrees n, b, $|\log(p^n)/\log(r^b) - 1| \leq 10^{-2}$ holds. This restriction on the field sizes is superficial and was chosen only to permit a fair comparison of the characteristics, and as such, these fields may not be directly of cryptographic interest. This is because in practice the group order $p^n - 1$ should possess a large prime factor dividing $\Phi_n(p)$, the n-th cyclotomic polynomial evaluated at p, to prevent the Pohlig-Hellman attack [85, 119] (cf. Chapter 2 for an explanation of, and Chapter 9 for a discussion of, this issue). For all three characteristics, we let d, the degree of the function field, vary from 3 to 7. The maximal degree of the sieve base elements was 70 for characteristic two, 60 for characteristic three, and 20 for characteristic 107. In none of the calculations we performed were these bounds reached, and so we can be confident that the time-optimal parameters we found in each case were optimal over all feasible parameters. The figures below compare the minimum expected running-times for the three characteristics, for each factor base size tested.

7.4.1 Discussion

We observe first that for each field size investigated, there appears to be a negligible difference between the difficulty of computing discrete logarithms in characteristics 2 and 3. For characteristic 107 however, the difference is significant. Furthermore, for the two smaller characteristics, the sieving time continues to fall

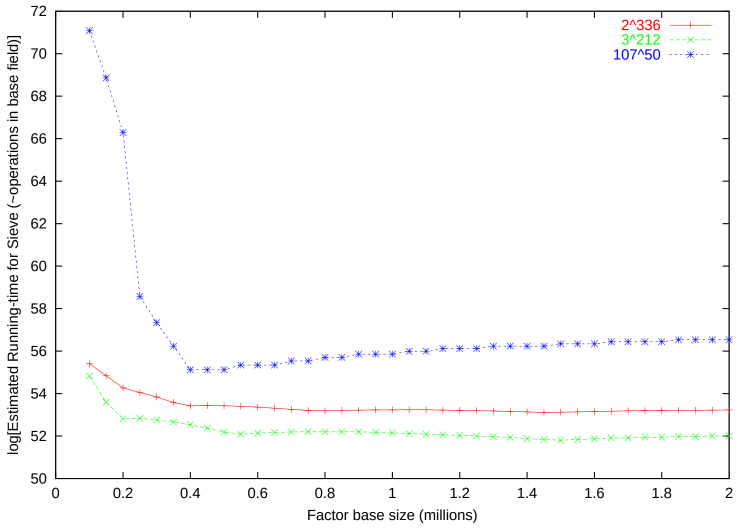


Figure 7.1: Minimum sieving times: 336 bits

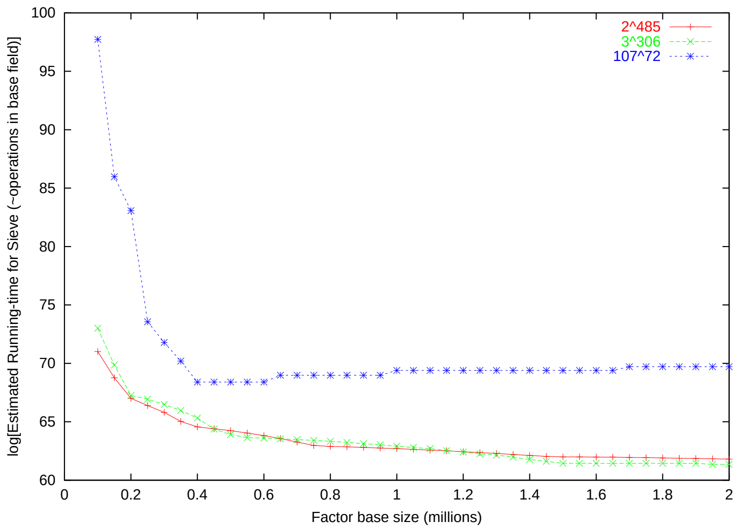


Figure 7.2: Minimum sieving times: 485 bits

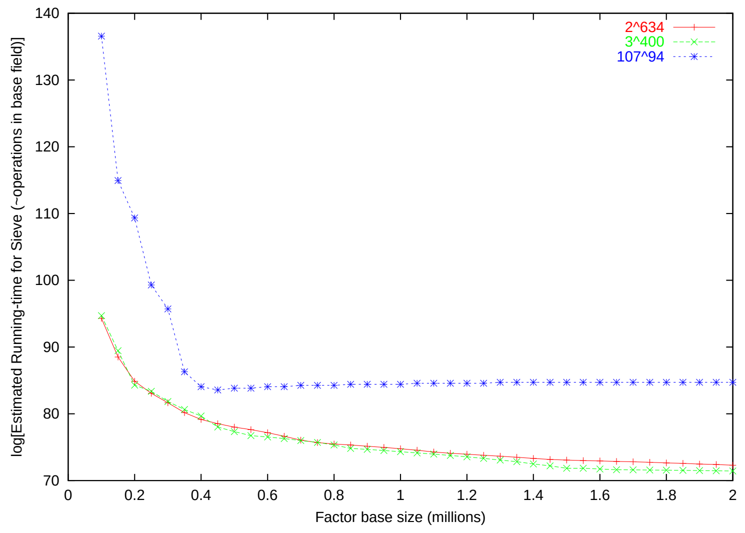


Figure 7.3: Minimum sieving times: 634 bits

quite smoothly as the factor base size is increased, whereas for $p = 107$, the optimum factor base size is reached for $\eta \approx 4.0 \times 10^5$, with the sieving time then remaining almost constant beyond this number. The asymptotic analysis of [3] applied to the field $\mathbb{F}_{107^{50}}$ implies that the optimum factor base size is about 4.0×10^6, and for the larger extension degrees is far larger than this still. Note however that for this p and n, this version of the FFS is an $L[1/2, c]$-algorithm, and so the results of [3] do not properly apply. Clearly though, if we are faced with this problem, we can not rely solely on asymptotic arguments if we are trying to optimise time-efficiency in using the FFS.

The question of whether or not increasing η improves the efficiency of a discrete logarithm computation can be seen from the figures to depend on the the field size, its characteristic, and the factor base size. The case $p = 107$ just mentioned shows that it is not necessarily beneficial to increase the factor base size as far as possible, though with the two smaller characteristics, we probably should increase η. However to make a fair assessment we need to analyse not only the sieving time but also the running-time for the linear algebra step. There are no references however which for sparse matrices give a precise complexity estimate, although it is commonly estimated to be $\mathcal{O}(\eta^2)$. We note though that the linear al-

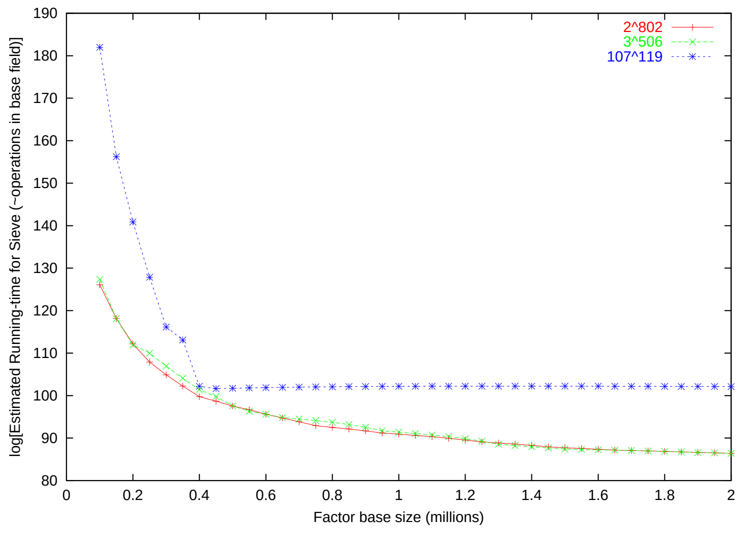


Figure 7.4: Minimum sieving times: 802 bits

gebra step for larger characteristics will in general be faster than for smaller ones; the degree of the relations generated among field elements is bounded by the extension degree of $\mathbb{F}_q$ over $\mathbb{F}_p$, and hence will on average be smaller in the former case, and so the corresponding matrix will be sparser and more easy to compute with. Moreover the probability of an element having a square factor is also diminished (it is $\approx 1/p$), and so most non-zero entries will be ± 1, further reducing the time needed for this step. The cut-off point at which the reward for increasing η becomes negligible will naturally depend on the implementation being used.

These results raise the obvious question of why in the fields $\mathbb{F}_{107^n}$ is the optimum factor base size seemingly much lower than that predicted by the asymptotic analysis, presuming it is adequate? To this end, we define a simple measure of the smoothness of a field in the usual representation. For a given factor base size η, let $S(\eta, \mathbb{F}_q^\times)$ be the proportion of η-smooth monic polynomials in $\mathbb{F}_q^\times$, where here η-smooth means the (m, α)-smoothness corresponding to a factor base of size η. The point of this measure is to obtain an idea of how increasing η affects the 'total' smoothness of a given field. In Figure 7.5 we show the computations of this measure for some η in our range for fields of bit-length 802 and characteristics 2, 3, 107, and also 199.

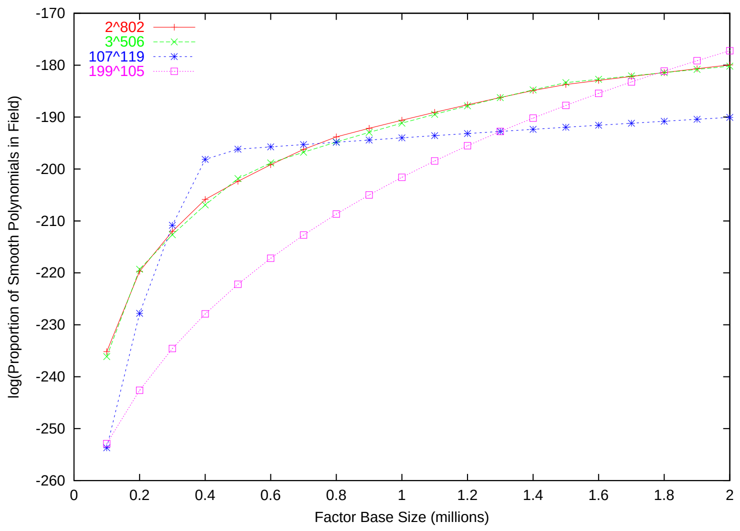


Figure 7.5: Smoothness measure $S(\eta, \mathbb{F}_{p^n}^{\times})$ for various 802-bit fields

We notice immediately that ignoring the characteristic 199 example, this graph is almost Figure 7.4 inverted. This is intuitive since the time needed for sieving is inversely related to the smoothness probabilities involved. It also suggests a reason why within the range of factor base sizes we ran estimates for, increasing this size did not yield a significant reduction in the running-time for $\mathbb{F}_{107^{119}}$: the total smoothness of $\mathbb{F}_{107^{119}}$ does not increase much for η from 4.0×10^5 to 2.0×10^6. Since this measure is independent of any particular implementation of the FFS, we can be fairly confident that for any index calculus computation there would be very little benefit in increasing the size of the factor base beyond 4.0×10^5 for this field, as a huge saving of time is gained in the linear algebra step.

In contrast, for $p = 199$ we found a significant reduction in our estimates for the minimum running-times over the same range of η. The improvement of running-time estimates for $\eta = 2.0 \times 10^6$ over $\eta = 4 \times 10^5$ for $p = 199$ is in fact by a factor of about 1.5×10^{15}. For $p = 2, 3,$ and 107, the corresponding improvements are 1.0×10^4, 3.2×10^4, and 1.1 respectively. The running-time estimates themselves for $\mathbb{F}_{199^{105}}$ are however still slower than those for $\mathbb{F}_{2^{802}}$ by a factor of 300 for $\eta = 2\times10^6$. Again, this may just reflect the improper application

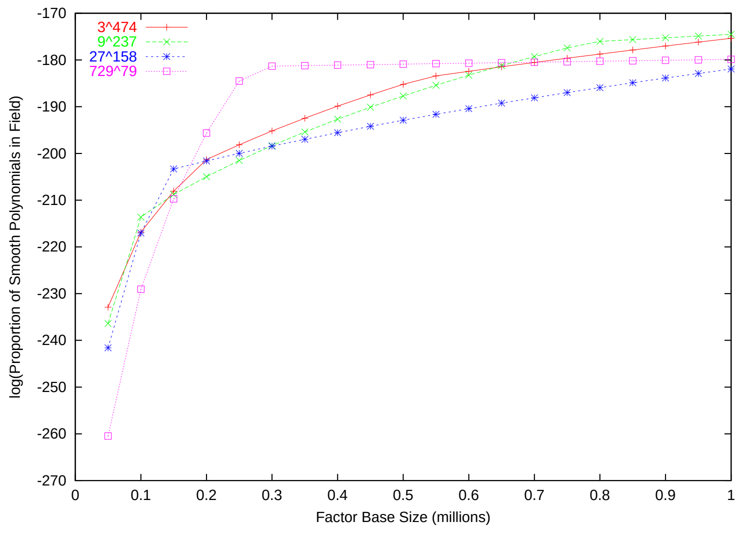


Figure 7.6: Comparison of $S(\eta, \mathbb{F}^{\times}_{3^{474}})$ for different field representations

of this version of the FFS to the DLP with these parameters.

Figure 7.5 indicates though that we may obtain useful information regarding the use of the FFS simply from this smoothness measure, in that while it does not take into account the details of any particular implementation the FFS, it may be used as a good heuristic for its behaviour.

Of course, given a system whose security is based on the DLP in a finite field, one can not choose the characteristic, and so these results may not seem applicable. However, for fields with a composite extension degree, we are free to choose which representation to work with, since every representation can be efficiently converted to any other [91, 92]. Figure 7.6 gives a comparison of $S(\eta)$ for four representations of the field $\mathbb{F}_{3^{474}}$. The graphs suggest that for fields with a composite extension degree, it may be the case that some representations are more susceptable to attack than others. Whether this translates into a noticible speed-up in practice remains to be seen. In the next chapter we present the results of an efficient implementation of the FFS in characteristic three to investigate these ideas.

7.5 Concluding Remarks

The purpose of this chapter was to gain an idea of how different parameter choices affect the performance of a basic probabilistic FFS model. The model we developed permits the easy computation of optimal parameter estimates that should be used in implementations, for any finite field where the FFS applies. This model may be developed or specialised to a particular implementation as needed.

Furthermore the question of whether finite fields with a composite extension degree are potentially weaker than others due to the multiple representations available was shown, with some heuristic evidence, to warrant further research. We present some research in this direction in the next chapter.

Chapter 8

Function Field Sieve in Characteristic Three

In this chapter we investigate the practical efficiency of the Function Field Sieve to compute discrete logarithms in finite fields of the form $\mathbb{F}_{3^{6m}}$, which arise from, and are essential to the security of, supersingular elliptic curves used in identity-based cryptography.

This chapter represents joint work with Andrew Holt, Dan Page, Nigel Smart and Fréderik Vercauteren, and appeared in [55].

8.1 Introduction

In light of the speculative results of Chapter 7, and motivated by attacks on identity based cryptosystems that use supersingular elliptic curves in characteristic three, in this chapter we report on the first implementation of the Function Field Sieve in this characteristic. Since the curves of interest have embedding degree six [43], we pay special attention to the fields $\mathbb{F}_{3^{6m}}$. A key observation is that this allows one to represent the function field over different base fields, which from the results presented in Chapter 7, leads one to expect differing performances for computing discrete logarithms.

Our practical experiments appear to show that a function field over $\mathbb{F}_3$ gives the best results, which is perhaps not too surprising. We show also that the exact analysis of Chapter 7 is more able to predict the behaviour, and thereby parameter choices, than the naive simple analysis. Finally, the results of this chapter give confidence that the key sizes one would use in a simple pairing-based system are likely to be secure against current algorithms and computing power.

8.2 The Function Field Sieve

The finite fields of interest to pairing based cryptography in characteristic three are given by $\mathbb{K} = \mathbb{F}_{3^n}$ where $n = 6 \cdot m$. In what follows we set $q = 3^m$. In practice, we limit ourselves to finite fields which could arise as the MOV embedding of a supersingular elliptic curve over the field $\mathbb{F}_q$ whose group order is divisible by a prime l of size comparable to that of q. Such elliptic curves have group orders given by

$$q \pm \sqrt{3q} + 1.$$

We wish to investigate not only the practicality of the function field sieve for the field extension $\mathbb{F}_{3^n}/\mathbb{F}_3$, but also the effect of taking different base fields, i.e., looking at the extension $\mathbb{F}_{3^n}/\mathbb{F}_{3^e}$ where $e = 1, 2, 3$ or 6. To this end we let $k = \mathbb{F}_{3^e}$, $N = n/e$ and $p = 3^e$.

We assume we are given $\alpha, \beta \in \mathbb{K}$, both of order l, such that

$$\beta = \alpha^x$$

for some unknown $x \in \{1, \ldots, l-1\}$. The discrete logarithm problem then is to compute x given α and β.

We will use a function field $F = k(X)[Y]/(H)$ defined by the polynomial $H(X,Y) \in k[X,Y]$ over the rational function field $k(X)$. Note, we shall abuse notation slightly and refer to the polynomial $H(X,Y)$ as the curve H, by which we mean the curve defined by $H(X,Y) = 0$. For practical reasons [96], one usually selects a C_{ab}-curve for H. However, in our examples we used a superelliptic curve of the form

$$H(X,Y) = Y^d + R(X)$$

where $R(X)$ is a polynomial in $k[X]$ of degree b. This enables more efficient calculation of the functions on the algebraic side, at the expense of a little less generality of our implementation. The class number of the function field F defined by H, or equivalently the number of points on the Jacobian of the curve J_H defined by H over k, we shall denote by $h = \#J_H(k)$.

We assume that the field $\mathbb{K}$ is defined by a polynomial basis with respect to the polynomial $f \in k[X]$ of degree N, i.e.,

$$\mathbb{K} \cong k[X]/(f).$$

To define the function field sieve algorithm we need to specify two polynomials $u_1, u_2 \in k[X]$ such that the norm of the function $u_1 + u_2 Y$, given by the resultant

$$u_2^d H(X, -u_1/u_2) = (-u_1)^d + u_2^d R$$

is divisible by f. In such a situation we have a surjective homomorphism

$$\phi : \left\{ \begin{array}{ccc} k[X,Y]/(H) & \longrightarrow & \mathbb{K} \cong k[X]/(f) \\ Y & \mapsto & -u_1/u_2. \end{array} \right.$$

We select a rational factor base $\mathcal{R}$ of small degree irreducible polynomials in $k[X]$ and an algebraic factor base $\mathcal{A}$ of small prime divisors in the divisor group $\mathrm{Div}(F)$. Hence, $\mathcal{R}$ and $\mathcal{A}$ are therefore defined by

$$\begin{aligned} \mathcal{R} &= \{\mathfrak{p} : \deg\, \mathfrak{p} \le B,\ \mathfrak{p} \text{ irreducible }\}, \\ \mathcal{A} &= \{\langle \mathfrak{p}, Y - \mathfrak{r}\rangle : \deg \mathfrak{p} \le B, \mathfrak{p} \text{ irreducible },\ \mathfrak{r} \equiv -u_1/u_2 \pmod{\mathfrak{p}}\}, \end{aligned}$$

for some smoothness bound B.

The goal of the Function Field Sieve is to find relatively prime pairs of polynomials (r, s), with $\deg r, \deg s \le l$, such that the polynomial $(su_2 - ru_1)$ and the divisor $\langle s+rY \rangle$ simultaneously factor over the respective factor bases, i.e.,

$$\begin{aligned} su_2 - ru_1 &= \prod_{\mathfrak{p}_i \in \mathcal{R}} \mathfrak{p}_i^{a_i} \\ \langle s+rY \rangle &= \sum_{\langle \mathfrak{p}_j, \mathfrak{r}_j \rangle \in \mathcal{A}} b_j \langle \mathfrak{p}_j, Y - \mathfrak{r}_j \rangle. \end{aligned}$$

Determining the factorization of $\langle s+rY \rangle$ over $\mathcal{A}$ is easily done by examining the factorization of the norm

$$N(\langle s+rY \rangle) = (-s)^d + r^d R.$$

Since $h \langle \mathfrak{p}_j, Y - \mathfrak{r}_j \rangle$ is a principal divisor, for each $\langle \mathfrak{p}_j, Y - \mathfrak{r}_j \rangle \in \mathcal{A}$ there exists a function $\lambda_j \in F^\times$ with $h \langle \mathfrak{p}_j, Y - \mathfrak{r}_j \rangle = \langle \lambda_j \rangle$ and such a function is unique up to multiplication by an element in $k^\times$. We then have that our algebraic relation is given by

$$(s+rY)^h = \mu \prod_{\lambda_j} \lambda_j^{b_j},$$

with μ an element in $k^\times$. We then apply the homomorphism ϕ above to obtain

$$(s - r\frac{u_1}{u_2})^h \equiv \prod_{\lambda_j} \phi(\lambda_j)^{b_j},$$

where $\equiv$ denotes equality modulo a possible factor in $k^\times$. Assuming h is coprime to $(p^N - 1)/(p-1)$, we can take h-th roots of both sides of this equation and if we write $\kappa_j = \phi(\lambda_j)^{1/h}$ we obtain

$$(s - r\frac{u_1}{u_2}) \equiv \prod_{\lambda_j} \kappa_j^{b_j}.$$

Combining the relation on the rational side with the above equation we find

$$\frac{1}{u_2} \prod_{\mathfrak{p}_i \in \mathcal{R}} \mathfrak{p}_i^{a_i} \equiv \prod_{\lambda_j} \kappa_j^{b_j}.$$

Hence, we obtain the relation between discrete logarithms given by

$$\sum_{\mathfrak{p}_i} a_i \log_g \mathfrak{p}_i - \log_g(u_2) = \sum_{\gamma_j} b_j \log_g \kappa_j, \tag{8.1}$$

where g is a multiplicative generator of the field $\mathbb{K}$. Note that we do not need at any point to compute the values of the κ_j, or for that matter the values of the λ_j, all that we require is that they exist, so that whilst we do compute the logarthims of the κ_j in the linear algebra elimination, the only logarithms we need are of those elements in the rational factor base. This is guaranteed by the condition that h should be coprime to $(p^N - 1)/(p - 1)$.

If sufficiently many independent relations of the form (8.1) have been obtained, we can solve for the discrete logarithms themselves using structured Gaussian elimination combined with the Lanczos method. Determining the discrete logarithm of β with respect to α is then performed using a standard recursive sieving strategy as explained in [131].

8.3 Choice of Parameters and Implementation Details

The various parameters of the function field sieve, namely the size d of the function field extension, the size B of the largest factor base element and the size l of the polynomials r and s, are approximated by a heuristic analysis [70] of the function field sieve as

$$\begin{aligned} l &= B, \\ B &= \left\lceil \left(\frac{4}{9}\right)^{1/3} N^{1/3} \log_p(N)^{2/3} \right\rceil, \\ d &= \left\lceil \sqrt{\frac{N}{B+1}} \right\rceil. \end{aligned}$$

Since we have restricted ourselves to superelliptic curves we need to ignore values of d which are divisible by three. However, in the range of our experiments this is not an issue and extending our results to the case where $d \equiv 0 \pmod 3$ can be accomplished using C_{ab}-curves [96], at the expense of more complicated formulae on the algebraic side.

8.3.1 Selection of f

Following Joux and Lercier [70] we first select u_1 and u_2 and then find a suitable value of f. We set $N' = N \pmod d$. However, since the degree $N = n/e$ may not be prime we split into three possible sub-cases:

- Case (i) : $\gcd(N', d) = 1$.
  We choose a curve H such that $R(X)$ is of degree $b = N'$ and such that $\#J_H(k)$ is coprime to $(p^N - 1)/(p-1)$. If this is not possible we go to Case (iii). We then select u_1 and u_2 of exact degrees $m-1$ and m respectively where $m = (N-b)/d$. The degree of the polynomial $f(x) = u_2^d H(X, -u_1/u_2)$ is then given by $\max\{d \cdot (m-1), d \cdot m + b\} = N$. Hence, we keep selecting u_1 and u_2 until f is irreducible.

- Case (ii) : $N' = 0$.
  We select $R(X)$ of smallest possible degree b such that $\#J_H(k)$ is coprime to $(p^N - 1)/(p-1)$. We then select u_1 and u_2 of exact degrees m and $m-1$ where $m = N/d$. The degree of the polynomial $f(x) = u_2^d H(X, -u_1/u_2)$ is then given by $\max\{d \cdot m, d \cdot (m-1) + b\} = N$, assuming $\deg R < d$. Hence, we keep selecting u_1 and u_2 until f is irreducible.

- Case (iii) : $\gcd(N', d) \neq 1, d$, or no suitable curve found above.
  Here we select $R(X)$ of degree $b < d$, with $\gcd(b, d) = 1$, such that $\#J_H(k)$ is coprime to $(p^N - 1)/(p-1)$ and for which one of t_1 and t_2 is minimal where

$$\begin{aligned} t_1 &= \max\{N, d \cdot m, d \cdot (m-1) + b\}, \\ t_2 &= \max\{N, d \cdot (m-1), d \cdot m + b\}. \end{aligned}$$

In the case of t_1 being minimal we then select u_1 and u_2 of degree m and $m-1$ until $u_2^d H(X, -u_1/u_2)$ if divisible by an irreducible polynomial f of degree N. In the case of t_2 being minimal we select u_1 and u_2 to be of degree $m-1$ and m, until we obtain an irreducible polynomial f of degree N which divides $u_2^d H(X, -u_1/u_2)$.

Note that Joux and Lercier [70] only considered Case (i) above, since they used arbitrary C_{ab}-curves (and not simply superelliptic ones), and because the extension degree N was always prime. Clearly, Case (iii) will lead to marginally less efficient relation collection, since the degree of the algebraic side will be slightly higher than if one was in Case (i) or (ii). However, if one is to deal with non-prime values of N and superelliptic curves one is led to such a case.

8.3.2 Lattice Sieving

The finding of (r, s) is performed using a lattice sieve. In the lattice sieve one selects a prime polynomial $\mathfrak{p} \in \mathcal{R}$ (resp. prime divisor $\langle \mathfrak{p}, Y - \mathfrak{r} \rangle \in \mathcal{A}$). Then one looks at the sub-lattice of the (r, s) plane such that $su_2 - ru_1$ (resp. $\langle s + rY \rangle$) is divisible by the chosen polynomial (resp. divisor). We found it more efficient to then sieve in the sub-lattice on a line-by-line basis, rather than sieve in a two-dimensional manner in the sub-lattice.

The use of lattice sieving has a number of advantages over sieving in the (r, s) plane. Firstly, it is better at yielding a large number of relations. Secondly, one can target factor base elements for which one does not yet have a relation. This enables one to obtain a matrix involving all elements in the factor base reasonably efficiently. Thirdly, the use of lattice sieving is crucial in the final stage where one wishes to target individual discrete logarithms using the recursive sieving strategy mentioned earlier.

8.3.3 Factor Base Size

A major consideration is the balancing of the effort required for the sieving and the matrix step. In theory one balances the two halves of the computation and so derives the complexity estimates such as those above. The size of both factor bases is approximated by p^B/B, however one should treat this size as a continuous function of a real variable B, as opposed to the discrete B above. See Chapter 7 for further analysis of this point.

However, sieving can be performed in a highly scalable manner. After all, to move from using a single computer performing the sieving to around 100-200 computers is relatively easy given the resources of most organisations these days. However, our code for the matrix step required the use of a single machine and hence does not scale.

Hence, in practice we can devote less total time to the matrix step compared to the sieving step. Since the matrix step has complexity approximately $O(T^2)$ where $T = \#\mathcal{R} + \#\mathcal{A} \approx 2p^B/B$, we see that we have a physical constraint on the size of the factor bases we can accommodate. In our experiments we assumed that solving a matrix with over half a million rows and columns was infeasible given our resources and matrix code. This sometimes led us to choose non-optimal, from a theoretical perspective, values for the other parameters.

8.3.4 Linear Algebra Step

Several studies involving use of index calculus-type methods discuss the linear algebra step, since it is a major practical bottleneck in the procedure. This is because parallelising existing algorithms is rather difficult. Various authors [82, 121], have identified several effective techniques for the solution of sparse linear systems modulo a prime. These include iterative schemes such as the Lanczos, Conjugate Gradient and Wiedemann algorithms, with or without a pre-processing step involving structured Gaussian elimination. Recently attention has moved onto attempting to perform this step in parallel, see for example [162] for the case of Lanczos over the field $\mathbb{F}_2$.

In our implementation we followed [82] and used a basic structured Gaussian elimination routine, such as that described in [8, 82, 121], so as to reduce the size of the linear system whilst maintaining a degree of sparsity. This submatrix is subsequently solved by the Lanczos algorithm [83], and a full result set is then recovered via back-substitution. Our implementation for this stage made use of Victor Shoup's NTL C++ library for multiprecision integers [143].

8.4 Experiments

As mentioned in Section 8.1, we are not only interested in how efficient the FFS method is on fields of relevance to IBE based systems. We are also interested in the effect of choosing a different base field $k = \mathbb{F}_{3^e}$ in the FFS. Usually for traditional discrete logarithm systems this would not be an issue since the extension degree is often prime. One should note that the theoretical analysis of Chapter 7 of the FFS shows that for a fixed field size the effect of the size of the base field is not as one would expect. This strange effect of the base field size 3^e on the overall performance for a fixed field size 3^n can be explained due to the non-continuous behaviour of various parameters, in particular the function field extension degree d. In the experiments below we selected field sizes $q = 3^n$ which arise from supersingular elliptic curves with group orders which are "almost" prime.

8.4.1 Field Size 3^{186}

This corresponds to a field size of approximately 295 bits. The rough theoretical estimates, given above, for the various values of $e = 1, 2, 3, 6$ are in the table below. A more careful analysis as in Chapter 7 reveals the following estimates, where for the factor bases we take the first $T/2$ primes (resp. $T/2$ prime places) on the rational (resp. algebraic side), in other words the value of B is not used directly.

	Rough Analysis			Analysis of Chapter 7	
e	d	B	$T \approx$	d	$T \approx$
1	4	12	89000	5	85000
2	4	6	180000	4	75000
3	4	4	270000	4	150000
6	3	2	530000	4	330000

We compared these results to the yields provided by our implementation and found that the best possible values seemed to be given by

e	d	$T \approx$
1	5	70000
2	4	90000
3	4	190000

This latter table was produced by comparing the yields of the implementation over a fixed time period for various parameter sizes and then selecting the one which would produce a full matrix in the shortest time period. We were however unable to generate suitable experimental data for the case $e = 6$ since this field is really too large to apply the FFS method in practice for such a value of n.

8.4.2 Field Size 3^{222}

This corresponds to a field size of approximately 352 bits. The rough theoretical estimates, given above, for the various values of $e = 1, 2, 3, 6$ along with the more precise estimates of Chapter 7, are in the following table.

	Rough Analysis			Analysis of Chapter 7	
e	d	B	$T \approx$	d	$T \approx$
1	4	13	250000	5	130000
2	4	6	180000	4	190000
3	4	4	270000	4	270000
6	4	2	530000	4	460000

We compared these results to the yields provided by our implementation and found that the best possible values seemed to be given by

e	d	$T \approx$
1	5	100000
2	4	190000
3	4	320000

Again the values for $e = 6$ are not given since n is still too small for this base field size to apply the FFS method successfully.

8.4.3 Field Size 3^{582}

We present this field since it is the first one which is usable in pairing based systems and which "could" be secure against current computing power on both the elliptic curve and the finite field sides. It corresponds to a bit length of 923 bits. The rough theoretical estimates are given below which should be compared to the the analysis in Chapter 7.

	Rough Analysis			Analysis of Chapter 7	
e	d	B	$T \approx$	d	$T \approx$
1	5	21	$9.0 \cdot 10^8$	6	$3.0 \cdot 10^9$
2	5	10	$7.0 \cdot 10^8$	7	$3.0 \cdot 10^9$
3	5	6	$1.0 \cdot 10^8$	7	$2.8 \cdot 10^9$
6	5	3	$2.0 \cdot 10^8$	5	$4.0 \cdot 10^8$

Note that these parameters would imply that such key sizes are currently out of range of the FFS method.

To completely confirm both our theoretical estimates and our partial experiments we ran a few experiments through to the final matrix stage and computation of individual logarithms.

Our experiments were run on a network of around 150 Unix based workstations. The network contained a number of older Sparc 5s and 10s running Solaris, plus a large number of Linux based machines with AMD1600 or Pentium 4 processors. We only used idle cycles of the machines whilst other people were using them, hence during the day we had the equivalent of 50 Pentium 4 machines working flat out. At night this increased to around 100 Pentium 4 machines. The matrix step was run on a Sun Blade 1000 workstation.

In Figure 8.1 we present the wall clock time t_1 needed to produce the relations, the time t_2 needed to solve the matrix step (divided into the time t_2' needed to perform the structured Gaussian elimination and the time t_2'' to solve the reduced system). We ran the sieving clients for time t_1 producing a total of R relations on T_a elements of the factor base. For many examples we did not try to find relations on all the factor base elements, since finding the relations on the last few elements

Figure 8.1: Timings for full discrete logarithm computations

n	186	186	186	222	222
N	186	93	62	222	111
e	1	2	3	1	2
d	5	4	4	5	4
$R(X)$	X	X	X	X^3+2X+1	X^3+X
T	70000	80000	140000	100000	160000
R	80045	96365	139376	96956	181675
T_a	69345	76425	137750	95169	158952
m	12634	13218	24746	20719	32148
t_1	8h	7h	30h	48h	50h
t_2'	43m	1h	2h 24m	3h	4h 20m
t_2''	13h	16h	1d 18h	2d 21h	4d 14h

can take a disproportionate amount of time. The line m denotes the approximate row-size of the resulting (approximately square) matrix after the application of structured Gaussian elimination. Note that our times for the linear algebra step can be considerably improved, using the techniques in [116].

8.5 Concluding Remarks

In this chapter we have reported on the first implementation of the FFS in characteristic three. We have paid particular attention to the case of finite fields which arise in pairing based cryptosystems. In particular such fields are of a composite nature and we have seen that this provides at best a marginal benefit in allowing one to apply the function field sieve over either $\mathbb{F}_3$ or $\mathbb{F}_{3^2}$: one should therefore simply use $\mathbb{F}_3$ as one's base field. This implies that fields of the form $\mathbb{F}_{3^{6m}}$ are, subject to this version of the FFS, as secure as characteristic three fields of a similar size but with prime extension degree, for example. The results of the following chapter however put the assumption of equivavlent security of these fields, subject to another attack, into question.

We have also shown that the exact analysis of Chapter 7 is more able to predict the behaviour, and thereby parameter choices, than a naive simple analysis. We have also shown how the key sizes one would use in a simple pairing based system are likely to be secure against current algorithms and computing power. However, the security of these fields, relative to charcteristic two, or large prime characteristic fields, still requires futher research.

We hope that our work will encourage others to investigate discrete logarithm algorithms in composite fields of characteristic three, and thereby allow the community to have greater faith in the security of pairing based systems which are based over such fields.

Chapter 9

On the Discrete Logarithm Problem on Algebraic Tori

In this chapter we propose the first index calculus algorithm for solving the DLP on Algebraic Tori, and apply this to the scheme presented in Chapter 5.

This chapter represents joint work with Fréderik Vercauteren, and appeared in [59]. The authors would like to thank Daniel Lazard for his invaluable comments regarding the details of the complexity of the Gröbner basis computation in the T_6-algorithm.

9.1 Introduction

Having developed efficient arithmetic for the implementation of torus-based cryptography in Part I of the thesis, we now focus on the security of these groups. The attack described in Chapters 7 and 8 solved the DLP in the field $\mathbb{F}_{q^n}$. In solving the DLP in this manner, one is also solving the DLP on the algebraic torus $T_n(\mathbb{F}_q)$, which is the group of real importance in the associated cryptosystems. By an easy argument, the security of $T_n(\mathbb{F}_q)$ and $\mathbb{F}_{q^n}^{\times}$ are equivalent (cf. Section 9.2).

There is thus the possibility that there exist attacks that apply directly to the torus, which conceivably may be more efficient than attacking the DLP via the field embedding. The relative size of the torus and the smallest field into which it embeds is $\phi(n)/n$, so one expects that if there is such an attack, then when this ratio is low, as it is for fields of interest in torus-based cryptography, the attack will be most efficient, relative to the full field attack. The situation is akin to attacking an elliptic curve via the MOV embedding [100]: if the embedding degree is high, then attacking the curve directly will be the more efficient option.

Until now, lacking any knowledge to the contrary, the security of $T_n(\mathbb{F}_q)$ has been based on two assumptions: firstly, $T_n(\mathbb{F}_q)$ should be large enough such that square root algorithms [101] are infeasible and secondly, the minimal finite field in which $T_n(\mathbb{F}_q)$ embeds should be large enough to thwart index calculus type attacks [101]. In these attacks one does not make any use of the particular form of the minimal surrounding finite field, i.e., $\mathbb{F}_{q^n}$, but only its size and the size of the subgroup of cryptographic interest.

In this chapter we develop an index calculus algorithm that works directly on rational tori $T_n(\mathbb{F}_q)$ and consequently show that the hardness of the DLP can depend on the form of the minimal surrounding finite field[1]. The algorithm is based on the purely algebraic index calculus approach by Gaudry [47] and exploits the compact representation of elements of rational tori. The very existence of such an algorithm shows that the lower communication cost offered by these tori, may also be exploited by the cryptanalyst.

In practice, the DLP in T_2 and T_6 are most important, since they determine the security of the cryptosystems LUC [146], XTR [88], CEILIDH [127], and pairing-based applications [43, 104]. We stress that when defined over prime fields $\mathbb{F}_p$, the security of these cryptosystems is not affected by our algorithm. Over extension fields however, this is not always the case. In this chapter, we provide a detailed description of our algorithm for $T_2(\mathbb{F}_{q^m})$ and $T_6(\mathbb{F}_{q^m})$. Note that this includes precisely the systems presented in [93], and also those described in [157] and Chapter 5 via the inclusion of $T_n(\mathbb{F}_p)$ in $T_2(\mathbb{F}_{p^{n/2}})$ and $T_6(F_{p^{n/6}})$ when n is divisible by two or six, respectively, which for efficiency reasons is always the case. Our method is fully exponential for fixed m and increasing q. From a complexity theoretic point of view, it is noteworthy that for certain very specific combina-

[1]Since the content of this chapter first appeared, there have been significant developments regarding the finite field DLP; see Section 9.6 for a short account of these.

tions of q and m, for example when $m! \approx q$, the algorithms run in expected time $L_{q^m}[1/2, c]$, which is comparable to the index calculus algorithm by Adleman and DeMarrais [2]. However, our focus will be on parameter ranges of practical cryptographic interest rather than asymptotic results.

A complexity analysis and prototype implementation of these algorithms, show that they are faster than Pollard-Rho in the *full* torus $T_2(\mathbb{F}_{q^m})$ for $m \geq 5$ and in the *full* torus $T_6(\mathbb{F}_{q^m})$ for $m \geq 3$. However, in cryptographic applications one would work in a prime order subgroup of $T_n(\mathbb{F}_{q^m})$ of order around 2^{160}; in this case, our algorithm is only faster than Pollard-Rho for larger m.

From a practical perspective, our experiments show that in the cryptographic range, the algorithm for $T_6(\mathbb{F}_{q^m})$ outperforms the corresponding algorithm for $T_2(\mathbb{F}_{q^{3m}})$ and that it is most efficient when $m = 4$ or $m = 5$. Furthermore, for $m = 5$, both algorithms in practice outperform Pollard-Rho in a subgroup of $T_6(\mathbb{F}_{q^5})$ of order 2^{160}, for q^{30} up to and including the 960-bit scheme based in $T_{30}(\mathbb{F}_p)$ proposed in Chapter 5. Compared to Pollard ρ our method seems to achieve in practice a 1000 fold speedup; its practical comparison with Adleman-DeMarrais is yet to be explored. Our experiments show that it is currently feasible to solve the DLP in $T_{30}(\mathbb{F}_p)$ with $\lceil \log_2 p \rceil = 20$, where we assume that a computation of around 2^{45} seconds total work effort is feasible.

The remainder of this chapter is organised as follows. In Section 9.2 we explicate the relation between the DLP in extension fields and the DLP in algebraic tori. In Section 9.3 we present the philosophy of our algorithm and explain how it is related to classical index calculus algorithms. In Sections 9.4 and 9.5 we give a detailed description of the algorithm for $T_2(\mathbb{F}_{q^m})$ and $T_6(\mathbb{F}_{q^m})$ respectively. Finally, we conclude and give pointers for further research in Section 9.6.

9.2 Discrete Logarithms in Extension Fields and Algebraic Tori

Extension fields possess a richer algebraic structure than prime fields, in particular those with highly composite extension degrees. This has led some researchers to suspect that such fields may be cryptographically weak. For instance, in 1984 Odlyzko stated that fields with a composite extension degree 'may be very weak' [113]. The main result of this chapter shows that these concerns may indeed be valid. As we show below, a naive attempt to exploit the available subfield structure of extension fields leads one to consider algebraic tori. While the following is perhaps trivial and is certainly well-known, we include it for expository purposes.

9.2.1 A Reduction of the DLP

Let $k = \mathbb{F}_q$ and let $K = \mathbb{F}_{q^n}$ be an extension of k of degree $n > 1$. Note that an element $\alpha \in K$ is actually in k if and only if $\varphi(\alpha) = \alpha$, where φ is the Frobenius automorphism, and recall from Chapter 2 that we have the norm map $N_{K/k} : K \to k$ given by

$$N_{K/k}(\alpha) = \alpha \cdot \varphi(\alpha) \cdots \varphi^{(n-2)}(\alpha) \cdot \varphi^{(n-1)}(\alpha) = \alpha^{(q^n-1)/(q-1)}.$$

Assume that $g \in K$ is a generator of $K^\times$ and let $h = g^s$ with $0 \le s < q^n - 1$ be an element we wish to find the discrete logarithm of with respect to g. The two classical approaches to this problem would be to use

- a combination of the Pohlig-Hellman reduction [119] to prime power order subgroups of $K^\times$ and a square root algorithm, such as Pollard-Rho [120],
- an index calculus algorithm in the full multiplicative group $K^\times$.

Note that the first approach has running time $O(\sqrt{p})$ with p the largest prime factor of $q^n - 1$, whereas the second has subexponential running time in q^n.

However, by using the norm map, it is possible to combine both approaches. Since g is a generator of $K^\times$, the norm $N_{K/k}(g)$ is a generator of $k^\times$. Taking norms then gives

$$N_{K/k}(h) = N_{K/k}(g)^{\bar{s}},$$

with $\bar{s} \equiv s \pmod{q-1}$. Furthermore, both $N_{K/k}(g)$ and $N_{K/k}(h)$ are elements of k, which allows us to use the index calculus approach in $k^\times$ instead of a square root algorithm.

If the extension degree n of K/k is composite, one can repeat the above norm argument with each intermediate subfield and obtain further congruency information regarding s, presuming one is able to solve the DLP in these subfields if necessary.

So how close to solving the DLP in $K^\times$ does this approach take us? For each proper divisor d of n, let k_d denote the degree d extension of k, so that $k \subseteq k_d \subsetneq K$. Assuming we can solve the DLP in each of the groups $k_d^\times$, the Chinese Remainder Theorem shows that we can compute s modulo

$$M = \operatorname{lcm}\{q^d - 1\}_{d|n, d\neq n} = \operatorname{lcm}\{\Phi_d(q)\}_{d|n,d\neq n}.$$

However, since the integers $\{\Phi_d(q)\}_{d|n,d\neq n}$ are not in general mutually coprime, it makes sense to define a function $f(n, q)$ as the quotient

$$f(n,q) = \frac{\prod_{d|n,d\neq n} \Phi_d(q)}{M}.$$

Using (2.1) then proves the following lemma.

Lemma 9.1. *By applying the norm with respect to each proper subfield $k_d \subsetneq K$ and presuming one can solve the DLP in each of these subfields, one can determine the discrete logarithm of an element $h \in \langle g\rangle$ with g a generator of $K^\times$ modulo*

$$M = \frac{q^n - 1}{\Phi_n(q) \cdot f(n,q)}.$$

The above lemma shows the existence of two obstructions to recover the full discrete logarithm. The first obstruction corresponds to the factor $f(n, q)$. Not much is known regarding the prime factorisation of the cyclotomic polynomials evaluated at prime powers, so it is difficult to obtain a good bound on the growth of $f(n, q)$ with increasing n. However, for n the product of the first k primes, which is the most important case in practice in torus based cryptography (cf. Section 2.2), the authors of [157] give an estimate for the size of the smallest integer $U(n, q)$ for which $\gcd(\Phi_d(q), \Phi_e(q), (q^n - 1)/U) = 1$ for all $d \neq e$ with $d \mid n$ and $e \mid n$. Lemma 7 of [157] states that $U = O(n^{2C})$, where $C \approx 0.374$ is Artin's constant, which measures the density of primes for which $p - 1$ is squarefree.

The relation with $f(n, q)$ is as follows: define $y_d = \gcd(\Phi_d(q), (q^n - 1)/U)$ for $d \mid n$, then by the definition of U, we have $\gcd(y_d, y_e) = 1$ for $e \neq d$ with $e \mid n$ and $d \mid n$. Again using the definition of U and $q^n - 1 = \prod_{d|n} \Phi_d(q)$, we conclude that $\prod_{d|n} y_d = (q^n - 1)/U$. Clearly $y_d \mid \Phi_d(q)$ and by the coprimality of the y_d we have

$$\prod_{d|n} y_d = \operatorname{lcm}\{y_d\}_{d|n} \;\mid\; \operatorname{lcm}\{\Phi_d(q)\}_{d|n}\,,$$

which implies that

$$q^n - 1 = \prod_{d|n} \phi_d(q) \;\mid\; U \cdot \operatorname{lcm}\{\Phi_d(q)\}_{d|n} \;\mid\; U \cdot \operatorname{lcm}\{\Phi_d(q)\}_{d|n, d \neq n} \cdot \Phi_n(q)\,.$$

By the definition of $f(n, q)$, we finally conclude that $f(n, q) \mid U(n, q)$ and in particular, $f(n, q) \leq U(n, q)$, which from our cryptanalytic perspective is negligible.

The second obstruction corresponds to solving the DLP in the subgroup of order $\Phi_n(q)$. Lacking an effective algorithm with which to solve the DLP in this subgroup, contemporary wisdom has simply presumed that it can be no easier than solving a DLP in $K^\times$ using the Number Field Sieve or Function Field Sieve algorithms, which exploit the ring structure of K, or using Pollard's rho method.

However, as observed by Rubin and Silverberg [127], the subgroup of order $\Phi_n(q)$ is isomorphic to the group of $\mathbb{F}_q$-rational points $T_n(\mathbb{F}_q)$ on an algebraic torus T_n (see Chapter 2). As a result, Rubin and Silverberg obtained bandwidth-efficient protocols. In this chapter, we analyse the destructive implications of this idea and obtain a new index calculus algorithm which works directly in the group $T_n(\mathbb{F}_q)$ instead of in the finite field $\mathbb{F}_{q^n}$.

In this section we showed that solving the DLP in $K = \mathbb{F}_{q^n}$ is almost equivalent (except for the factor $f(n, q)$) to solving the DLP in all proper subfields $k_d \subsetneq K$ and the torus $T_n(\mathbb{F}_q)$. By repeating this argument for the fields k_d, solving the DLP in $K = \mathbb{F}_{q^n}$ is actually almost equivalent to solving the DLP in the algebraic tori $T_d(\mathbb{F}_q)$ for $d \mid n$. The missing factor equals

$$\frac{q^n - 1}{\operatorname{lcm}\{\Phi_d(q)\}_{d|n}} \;\mid\; U(n,q)\,,$$

which again is negligible in the cryptographic setting. Hence finding an efficient algorithm to solve the DLP on algebraic tori enables one to solve DLPs in extension fields, as well as vice versa.

9.3 Algorithm Philosophy

The algorithm as presented in Sections 9.4 and 9.5 is based on an idea first proposed by Gaudry [47], which itself is generalisation of an idea of semaev [138], in reference to the DLP on general abelian varieties. While Gaudry's method is in principle an index calculus algorithm, the ingredients are very algebraic: for instance one need not rely on unique factorisation to obtain a notion of 'smoothness', as in finite field discrete logarithm algorithms.

As an introduction, in this section we consider Gaudry's idea in the context of computing discrete logarithms in $\mathbb{F}_{q^m}^{\times}$, and show how it is related to classical index calculus.

9.3.1 Classical Method

Let $\mathbb{F}_{q^m} = \mathbb{F}_q[t]/(f(t))$ for some monic irreducible degree m polynomial and let the basis be $\{1, t, \ldots, t^{m-1}\}$. Let g be a generator of $\mathbb{F}_{q^m}^{\times}$ and let $h \in \langle g \rangle$ be an element whose logarithm to base g is desired. Suppose also, for this example, that we are able to deal with a factor base of size q.

Classically, one would first reduce the problem to considering only monic polynomials, i.e., one considers the quotient $\mathbb{F}_{q^m}^{\times}/\mathbb{F}_q^{\times}$, and defines a factor base

$$\mathcal{F} = \{t + a : a \in \mathbb{F}_q\}.$$

Then for random $j, k \in \mathbb{Z}/((q^m - 1)/(q-1))\mathbb{Z}$ one computes $r = g^j h^k$ and tests whether $r/\mathrm{lc}(r)$ decomposes over $\mathcal{F}$, with $\mathrm{lc}(r)$ the leading coefficient of r. This occurs with probability approximately $1/(m-1)!$ for large q since the set of all products of $m-1$ elements of $\mathcal{F}$ generates roughly $q^{m-1}/(m-1)!$ elements of $\mathbb{F}_{q^m}^{\times}/\mathbb{F}_q^{\times}$.

Computing more than q such relations allows one to compute $\log_g h \bmod (q^m-1)/(q-1)$ as usual with a linear algebra elimination (and one applies the norm $N_{\mathbb{F}_{q^m}/\mathbb{F}_q}$ to g and h and solves the corresponding DLP in $\mathbb{F}_q^{\times}$ to recover the remaining modular information).

9.3.2 Gaudry's Method

Two essential points taken for granted in the above description are that there exist efficient procedures to compute:

- whether a given r decomposes over $\mathcal{F}$; this happens precisely when $r \in \mathbb{F}_q[t]$ splits over $\mathbb{F}_q$ or equivalently when $\gcd(t^q - t, r/\mathrm{lc}(r)) = r/\mathrm{lc}(r)$,
- the actual decomposition of r, i.e., to compute the roots of $r \in \mathbb{F}_q[t]$ in $\mathbb{F}_q$.

One may equivalently consider the following problem: determine whether the system of equations obtained by equating powers of t in the equality

$$\prod_{i=1}^{m-1}(t+a_i) = r/\mathrm{lc}(r) = r_0 + r_1 t + \cdots + r_{m-2}t^{m-2} + t^{m-1}, \tag{9.1}$$

has a solution $(a_1, \ldots, a_{m-1}) \in \mathbb{F}_q^{m-1}$ and if so, to compute one such solution. Of course, in this trivial example the roots a_i can be read off from the factorisation of $r/\mathrm{lc}(r)$. However, one obtains a non-trivial example if the group operation on the left is more sophisticated than polynomial multiplication, such as elliptic curve point addition, which was Gaudry's original motivation for developing the algorithm. In this case the decomposition of a group element over the factor base can become more sophisticated, but the principle remains the same.

The central benefit of this perspective is that it can be applied in the absence of unique factorisation, since with a suitable choice of factor base, or more accurately a decomposition base, one can simply induce relations algebraically. For example, approaching the above problem from this slightly different perspective gives an algorithm for working directly in $\mathbb{F}_{q^m}^\times$, which is perhaps more natural than the stated quotient, $\mathbb{F}_{q^m}^\times/\mathbb{F}_q^\times$. Define a decomposition base

$$\mathcal{F} = \{1 + at : a \in \mathbb{F}_q\},$$

and again associate to the equality

$$\prod_{i=1}^{m}(1 + a_i t) \equiv r \equiv r_0 + r_1 t + \cdots + r_{m-1}t^{m-1} \pmod{f(t)}, \tag{9.2}$$

the algebraic system obtained by equating powers of t.

Note that in (9.2) one must multiply m elements of $\mathcal{F}$ in order to obtain a probability of $1/m!$ for obtaining a relation, rather than the $m-1$ elements (and probability $1/(m-1)!$) of (9.1). The reason these probabilities differ is simply that the algebraic groups $\mathbb{F}_{q^m}^\times/\mathbb{F}_q^\times$ and $\mathbb{F}_{q^m}^\times$ over $\mathbb{F}_q$ are $m-1$ and m-dimensional respectively.

Ignoring for the moment that $\mathcal{F}$ essentially consists of degree one polynomials, and assuming that we want to solve this system without factoring $r/\text{lc}(r)$, we are faced with finding a solution to a non-linear system, which would ordinarily require a Gröbner basis computation to solve. However writing out the left hand side in the polynomial basis $\{1, \ldots, t^{m-1}\}$ gives

$$\begin{aligned}\prod_{i=1}^{m}(1 + a_i t) &= 1 + \overline{\sigma}_1 t + \cdots + \overline{\sigma}_m t^m \\ &\equiv 1 + \overline{\sigma}_1 t + \cdots + \overline{\sigma}_{m-1}t^{m-1} + \overline{\sigma}_m(t^m - f(t)) \pmod{f(t)},\end{aligned}$$

with $\overline{\sigma}_i$ the i-th elementary symmetric polynomial in the a_i. Equating powers of t then gives a linear system of equations in the $\overline{\sigma}_i$ for $i = 1, \ldots, m$. Given a solution $(\sigma_1, \ldots, \sigma_m)$ to this system of equations, r will decompose over $\mathcal{F}$ precisely when the polynomial

$$p(x) := x^m - \sigma_1 x^{m-1} + \sigma_2 x^{m-2} - \cdots + (-1)^m \sigma_m$$

splits over $\mathbb{F}_q$. Thus exploiting the symmetry in the construction of the algebraic system makes solving it much simpler. Although in this contrived example, solving the system directly and solving it using its symmetry are essentially the same, in general the latter makes infeasible computations feasible.

Following from this example, a simple observation is that for an algebraic group over $\mathbb{F}_q$ whose representation is m-dimensional, then using a decomposition base $\mathcal{F}$ of q elements, one must add m elements of $\mathcal{F}$ to obtain a constant probability of decomposition $1/m!$. Therefore, we conclude that the more efficient the representation of the group is, the higher the probability of obtaining a relation, and thus the corresponding index calculus algorithm will be more efficient.

In the following two sections, we apply this idea to rational representations of algebraic tori, and show that the above probability of $1/m!$ can be reduced significantly to $1/(m/2)!$ when m is divisible by 2 and to $1/(m/3)!$ when m is divisible by 6.

9.4 An Index Calculus Algorithm for $T_2(\mathbb{F}_{q^m}) \subset \mathbb{F}_{q^{2m}}^\times$

For q any odd prime power, we describe an algorithm to compute discrete logarithms in $T_2(\mathbb{F}_{q^m})$.

9.4.1 Setup

With regard to the extension $\mathbb{F}_{q^{2m}}/\mathbb{F}_{q^m}$, by Lemma 2.1 we know that

$$\#T_2(\mathbb{F}_{q^m}) = \Phi_2(q^m) = q^m + 1,$$

and hence we presume the DLP we consider is in the subgroup of this order. By applying the reduction of the DLP via norms as in Section 9.2, it is clear that the hard part actually is $T_{2m}(\mathbb{F}_q) \subsetneq T_2(\mathbb{F}_{q^m})$. Since in this section we use the properties of T_2 rather than T_{2m}, we only consider $T_2(\mathbb{F}_{q^m})$, or more accurately $(\mathrm{Res}_{\mathbb{F}_{q^m}/\mathbb{F}_q} T_2)(\mathbb{F}_q)$, where here Res denotes the Weil restriction of scalars (see for example [128] or [161]).

Let $\mathbb{F}_{q^m} \cong \mathbb{F}_q[t]/(f(t))$ with $f(t) \in \mathbb{F}_q[t]$ an irreducible monic polynomial of degree m and take the polynomial basis $\{1, t, \ldots, t^{m-1}\}$. Assuming that q is an odd prime power, we let $\mathbb{F}_{q^{2m}} = \mathbb{F}_{q^m}[\gamma]/(\gamma^2 - \delta)$ with basis $\{1, \gamma\}$, for some non-square $\delta \in \mathbb{F}_{q^m} \setminus \mathbb{F}_q$. Then using Definition 2.5, we see that

$$T_2(\mathbb{F}_{q^m}) = \{(x, y) \in \mathbb{F}_{q^m} \times \mathbb{F}_{q^m} : x^2 - \delta y^2 = 1\}.$$

This representation uses two elements of $\mathbb{F}_{q^m}$ to represent each point. The torus T_2 is one-dimensional, rational, and has the following equivalent affine representation:

$$T_2(\mathbb{F}_{q^m}) = \left\{ \frac{z - \gamma}{z + \gamma} : z \in \mathbb{F}_{q^m} \right\} \cup \{O\}, \tag{9.3}$$

where O is the point at infinity.

Here a point $g = g_0 + g_1\gamma \in T_2(\mathbb{F}_{q^m})$ in the $\mathbb{F}_{q^{2m}}$ representation has a corresponding representation as given above by the rational function $z = -(1 + g_0)/g_1$ if $g_1 \neq 0$, whilst the elements -1 and 1 map to $z = 0$ and $z = O$ respectively. The representation (9.3) thus gives a compression factor of two for the elements of $\mathbb{F}_{q^{2m}}$ that lie in $T_2(\mathbb{F}_{q^m})$. Furthermore since $T_2(\mathbb{F}_{q^m})$ has $q^m + 1$ elements, this compression is optimal (since for this example, including the point at infinity, we really have a map from $T_2(\mathbb{F}_{q^m}) \to \mathbb{P}^1(\mathbb{F}_{q^m})$).

9.4.2 Decomposition Base

As with any index calculus algorithm, we need to define a factor base, or in the case of Gaudry's algorithm, a decomposition base. Let

$$\mathcal{F} = \left\{ \frac{a-\gamma}{a+\gamma} : a \in \mathbb{F}_q \right\} \subset T_2(\mathbb{F}_{q^m}), \tag{9.4}$$

which contains q elements, since the map, given above, is a birational isomorphism from T_2 to $\mathbb{A}^1$. Note that if $\delta \in \mathbb{F}_q$, then $\mathcal{F}$ would lie in the subvariety $T_2(\mathbb{F}_q)$ and would not aid in our attack, which is why we ensured that $\delta \in \mathbb{F}_{q^m} \setminus \mathbb{F}_q$ during the setup.

9.4.3 Relation Finding

Writing the group operation multiplcatively, let P be a generator, and let $Q \in \langle P \rangle$ be a point we wish to find the discrete logarithm of with respect to P. For a given $R = P^j \cdot Q^k$, we test whether it decomposes as a product of m points in the decomposition base:

$$\prod_{i=1}^{m} P_i = R, \tag{9.5}$$

with $P_1, \ldots, P_m \in \mathcal{F}$. From the representation we have chosen for T_2 we may equivalently write this as

$$\prod_{i=1}^{m} \left(\frac{a_i - \gamma}{a_i + \gamma} \right) = \frac{r-\gamma}{r+\gamma},$$

where the a_i are unknown elements in $\mathbb{F}_q$, and $r \in \mathbb{F}_{q^m}$ is the affine representation of R. Note that the left hand side is symmetric in the a_i. Upon expanding the product for both the numerator and denominator, we obtain two polynomials of degree m in γ whose coefficients are just plus or minus the elementary symmetric polynomials $\sigma_i(a_1, \ldots, a_m)$ of the a_i:

$$\frac{\sigma_m - \sigma_{m-1}\gamma + \cdots + (-1)^m \gamma^m}{\sigma_m + \sigma_{m-1}\gamma + \cdots + \gamma^m} = \frac{r-\gamma}{r+\gamma}.$$

Therefore, when we reduce modulo the defining polynomial of γ, we obtain an equation of the form

$$\frac{b_0(\sigma_1,\ldots,\sigma_m) - b_1(\sigma_1,\ldots,\sigma_m)\gamma}{b_0(\sigma_1,\ldots,\sigma_m) + b_1(\sigma_1,\ldots,\sigma_m)\gamma} = \frac{r-\gamma}{r+\gamma},$$

where b_0, b_1 are linear in the σ_i and have coefficients in $\mathbb{F}_{q^m}$. More explicitly, since $\gamma^2 = \delta \in \mathbb{F}_{q^m}$, these polynomials are given by

$$b_0 = \sum_{k=0}^{\lfloor m/2 \rfloor} \sigma_{m-2k}\delta^k \quad \text{and} \quad b_1 = \sum_{k=0}^{\lfloor (m-1)/2 \rfloor} \sigma_{m-2k-1}\delta^k\,,$$

where we define $\sigma_0 = 1$.

In order to obtain a simple set of algebraic equations amongst the σ_i, we first reduce the left hand side to the affine representation (9.3) and obtain the equation

$$b_0(\sigma_1,\ldots,\sigma_m) - b_1(\sigma_1,\ldots,\sigma_m)r = 0.$$

Since the unknowns σ_i are elements of $\mathbb{F}_q$, we express the above equation on the polynomial basis of $\mathbb{F}_{q^m}$ to obtain m linear equations over $\mathbb{F}_q$ in the m unknowns $\sigma_i \in \mathbb{F}_q$. This gives an $m \times m$ matrix M over $\mathbb{F}_q$ such that

- the $(m-2k)$-th column contains the coefficients of δ^k,
- the $(m-2k-1)$-th column contains the coefficients of $-r\delta^k$.

Furthermore, let V be the $m \times 1$ vector containing the coefficients of $r\delta^{(m-1)/2}$ when m is odd or $-\delta^{m/2}$ when m is even, then $\Sigma = (\sigma_1,\ldots,\sigma_m)^T$ is a solution of the linear system of equations

$$M\Sigma = V\,.$$

If there is a solution Σ, to see whether this corresponds to a solution of (9.5) we test whether the polynomial

$$p(x) := x^m - \sigma_1 x^{m-1} + \sigma_2 x^{m-2} - \cdots + (-1)^m \sigma_m$$

splits over $\mathbb{F}_q$ by computing $g(x) := \gcd(x^q - x, p(x))$. If $g(x) = p(x)$, then the roots $a_1, \dots, a_m$ will be the affine representation of the elements of the factor base which sum to R and we have found a relation.

9.4.4 Complexity Analysis and Experiments

The number of elements of $T_2(\mathbb{F}_{q^m})$ generated by all sums of m points in $\mathcal{F}$ is roughly $q^m/m!$, assuming no repeated summands and that most points admit a unique factorisation over the factor base. Hence the probability of obtaining a relation is approximately $1/m!$. Therefore in order to obtain q relations we must perform roughly $m!q$ such decompositions. Each decomposition consists of the following steps:

- computing the matrix M and vector V takes $O(m^3)$ operations in $\mathbb{F}_q$, using a naive multiplication routine,
- solving for Σ also requires $O(m^3)$ operations in $\mathbb{F}_q$,
- computing the polynomial $g(x)$ requires $O(m^2 \log q)$ operations in $\mathbb{F}_q$,
- if the polynomial $p(x)$ splits over $\mathbb{F}_q$, then we have to find the roots $a_1, \dots, a_m$ which requires $O(m^2 \log m(\log q + \log m))$ operations in $\mathbb{F}_q$.

Note that the last step only has to be executed $O(q)$ times. The overall complexity to find $O(q)$ relations is therefore

$$O(m! \cdot q \cdot (m^3 + m^2 \log q)) .$$

operations in $\mathbb{F}_q$.

Since in each row of the final relations matrix there will be $O(m)$ non-zero elements, we conclude that finding a kernel vector using sparse matrix techniques [82] requires $O(mq^2)$ operations in $\mathbb{Z}/(q^m + 1)\mathbb{Z}$ or about $O(m^3q^2)$ operations in $\mathbb{F}_q$. This proves the following theorem.

Theorem 9.1. *The expected running time of the T_2-algorithm to compute discrete logarithms in $T_2(\mathbb{F}_{q^m})$ is*

$$O(m! \cdot q \cdot (m^3 + m^2 \log q) + m^3 q^2)$$

operations in $\mathbb{F}_q$.

Note that when $m > 1$ and the q^2 term dominates, by reducing the size of the decomposition base, the complexity may be reduced to $O(q^{2-2/m})$ for $q \to \infty$ using the results of Thériault [154], and a refinement reported independently by Gaudry *et al.* [49] and Nagao [106].

The expected running time of the T_2-algorithm is minimal when the relation stage and the linear algebra stage take comparable time, i.e., when $m! \cdot q \cdot (m^3 + m^2 \log q) \simeq m^3 q^2$ or $m! \simeq q$. The complexity of the algorithm then becomes $O(m^3 q^2)$, which can be rewritten as

$$\begin{aligned} O(m^3 q^2) &= O\big(\exp(3 \log m + 2 \log q)\big) \\ &= O\big(\exp(5 (\log q)^{1/2} (\log q)^{1/2})\big) \\ &= O\big(\exp(5 (m \log m)^{1/2} (\log q)^{1/2})\big) \\ &= O\big(L_{q^m}(1/2, c)\big) \end{aligned}$$

with $c \in \mathbb{R}_{>0}$. Note that for the first equality we have used $\log m < \log q$, and for the second and third equality we have used that $m! \simeq q$, and thus by taking logarithms $\log q \simeq m \log m$.

To assess the practicality of the T_2 algorithm, we ran several experiments using a simple Magma implementation, the results of which are given in Figure 9.1. This table should be read as follows: the size of the torus cardinality, i.e., $\log_2(q^m)$, is constant across each row; for a given q^m, the table contains for $m = 1, \ldots, 15$, the $\log_2$ of the expected running times in seconds for the entire algorithm, i.e., both relation collection stage and linear algebra. For instance, for $q^m \cong 2^{300}$ and $m = 15$, the total time would be approximately 2^{51} seconds on one AMD 1700+ using our Magma implementation. For the fields where the torus is less than 160 bits in size, we use the full torus otherwise we use a subgroup of 160 bits

Figure 9.1: $\log_2$ of expected running times (s) of the T_2-algorithm and Pollard-Rho in a subgroup of size 2^{160}: bold for time $< 2^{45}$ and matrix of size $< 2^{23}$.

			m						
$\log_2 \lvert\mathbb{F}_{q^{2m}}\rvert$	$\log_2 \lvert T_2(\mathbb{F}_{q^m})\rvert$	ρ	1	2	3	4	5	6	7
200	100	34	88	40	52	36	**26**	**20**	**16**
300	150	59	138	66	87	62	48	38	**31**
400	200	65	188	92	121	88	68	55	46
500	250	66	238	117	155	114	89	73	61
600	300	66	289	142	189	139	110	90	76
700	350	66	339	168	223	165	130	107	91
800	400	66	389	193	256	190	150	124	105
900	450	68	439	219	290	215	171	141	120
1000	500	69	489	244	324	241	191	158	134

			m							
$\log_2 \lvert\mathbb{F}_{q^{2m}}\rvert$	$\log_2 \lvert T_2(\mathbb{F}_{q^m})\rvert$	ρ	8	9	10	11	12	13	14	15
200	100	34	**17**	**18**	**21**	**23**	**26**	**31**	**33**	**37**
300	150	59	**26**	**25**	**26**	**28**	**31**	**34**	**37**	**40**
400	200	65	39	**34**	**32**	**33**	**35**	**38**	**41**	**44**
500	250	66	52	45	40	**38**	**40**	**42**	**44**	47
600	300	66	65	57	51	45	44	46	48	51
700	350	66	78	69	61	55	50	50	52	54
800	400	66	91	80	71	64	58	56	55	58
900	450	68	104	92	82	74	67	62	61	62
1000	500	69	117	103	92	83	76	69	66	67

to estimate the Pollard ρ costs (cf. Section 9.4.5 for a full explanation).

Note that Figure 9.1 does not take into account *memory* constraints imposed by the linear algebra step; since the number of relations is approximately q, we conclude that the algorithm is currently only practical for $q \leq 2^{23}$. Assuming that 2^{45} seconds total work effort, which is about 1.1×10^6 years, is feasible and assuming it is possible to find a kernel vector of a sparse matrix of dimension 2^{23}, Table 9.1 contains, in bold, the combinations of q and m which can be handled using our Magma implementation.

9.4.5 Comparison with other Methods

In this section we compare the T_2-algorithm with the Pollard-Rho and the Adleman-DeMarrais index calculus algorithm.

Pollard-Rho in the Full Torus

Using the Pohlig-Hellman reduction, the overall running time is determined by executing the Pollard-Rho algorithm in the subgroup of $T_2(q^m)$ of largest prime order l. Since $\#T_2(q^m) = q^m + 1$, we have to analyse the size of the largest prime factor l. Note that the factorisation of $x^m + 1$ over $\mathbb{Z}[x]$ is given by

$$x^m + 1 = \frac{x^{2m} - 1}{x^m - 1} = \frac{\prod_{d|2m} \Phi_d(x)}{\prod_{d|m} \Phi_d(x)} = \prod_{d|2m, d\nmid m} \Phi_d(x)\,,$$

which implies that the maximum size of the prime l is $O(q^{\phi(2m)})$, since the degree of $\Phi_{2m}(x)$ is $\phi(2m)$. The overall worst case complexity of this method is therefore $O(q^{\phi(2m)/2})$ operations in $\mathbb{F}_{q^{2m}}$ or $O(m^2 \cdot q^{\phi(2m)/2})$ operations in $\mathbb{F}_q$.

From a complexity theoretic point of view, we therefore conclude that for $m! \leq q$, our algorithm is as fast as Pollard-Rho whenever $m \geq 5$, since then $\phi(2m)/2 > 2$. As a consequence, we note that the T_2 algorithm does not lead to an improvement over existing attacks on LUC [146], XTR [88] or CEILIDH [127] over $\mathbb{F}_p$. Furthermore, also the security of MNT curves [104] defined over $\mathbb{F}_p$, where p is a large prime remains unaffected.

Pollard-Rho in a Subgroup of prime order $\simeq 2^{160}$

In cryptographic applications however, one would work in a subgroup of $T_2(\mathbb{F}_{q^m})$ of prime order l with $l \simeq 2^{160}$. To this end, we measured the average time taken for one multiplication for the various fields in Magma, and multiplied this time by the expected 2^{80} operations required by the Pollard-Rho algorithm. The results can be found in the third column of Table 9.1. The column for $m = 15$ is especially interesting since this determines the security of the T_{30} cryptosystem introduced in

Chapter 5. In this case, the T_2 is always faster than Pollard-Rho, and the matrices occurring in the linear algebra step would be feasible up to 700-bit fields.

Remark 9.1. *The linearity of the decomposition method in fact holds for any torus T_{p^r}, where by this we mean that to solve for the values of the symmetric polynomials σ_i, one only needs to use linear algebra. However, amongst all such tori the most efficient decomposition is for T_{2^r}, since $p^r/\phi(p^r)$ is maximal in this case, and hence the probability of obtaining a relation is maximised. When one considers T_n for which n is divisible by more than one distinct prime factor, the rational parametrisation becomes non-linear, and hence so does the corresponding decomposition, as we see in the following section.*

9.5 An Index Calculus Algorithm for $T_6(\mathbb{F}_{q^m}) \subset \mathbb{F}_{q^{6m}}^\times$

In this section we detail our algorithm to compute discrete logarithms in $T_6(\mathbb{F}_{q^m})$. The main difference with the T_2-algorithm is the non-linearity of the equations involved in the decomposition step.

9.5.1 Setup

Again, let $\mathbb{F}_{q^m} \cong \mathbb{F}_q[t]/(f(t))$, with $f(t)$ an irreducible polynomial of degree m and where we use the polynomial basis $\{1, t, t^2, \ldots, t^{m-1}\}$. Since T_6 is two-dimensional and rational, it is an easy exercise (see Section 2.2.3) to construct a birational map from T_6 to $\mathbb{A}^2$ for a given representation of $\mathbb{F}_{q^{6m}}$. For the following exposition we make use of the the CEILIDH field representation and maps, as described in [127].

Let $q^m \equiv 2$ or $5 \bmod 9$, and for $(r, q) = 1$ let ζ_r denote a primitive r-th root of unity over $\mathbb{F}_{q^m}$. Define $x = \zeta_3$ and let $y = \zeta_9 + \zeta_9^{-1}$, then clearly $x^2 + x + 1 = 0$ and $y^3 - 3y + 1 = 0$. Let $\mathbb{F}_{q^{3m}} = \mathbb{F}_{q^m}(y)$ and $\mathbb{F}_{q^{6m}} = \mathbb{F}_{q^{3m}}(x)$, then the bases we use are $\{1, y, y^2 - 2\}$ for the degree three extension and $\{1, x\}$ for the degree two extension.

Let $V(f)$ be the zero set of $f(\alpha_1, \alpha_2) = 1 - \alpha_1^2 - \alpha_2^2 + \alpha_1\alpha_2$ in $\mathbb{A}^2(\mathbb{F}_{q^m})$, then we have the following inverse birational maps:

- $\psi : \mathbb{A}^2(\mathbb{F}_{q^m}) \setminus V(f) \xrightarrow{\sim} T_6(\mathbb{F}_{q^m}) \setminus \{1, x^2\}$, defined by

$$\psi(\alpha_1, \alpha_2) = \frac{1 + \alpha_1 y + \alpha_2(y^2 - 2) + (1 - \alpha_1^2 - \alpha_2^2 + \alpha_1\alpha_2)x}{1 + \alpha_1 y + \alpha_2(y^2 - 2) + (1 - \alpha_1^2 - \alpha_2^2 + \alpha_1\alpha_2)x^2}, \tag{9.6}$$

- $\rho : T_6(\mathbb{F}_{q^m}) \setminus \{1, x^2\} \xrightarrow{\sim} \mathbb{A}^2(\mathbb{F}_{q^m}) \setminus V(f)$, which is defined as follows: for $\beta = \beta_1 + \beta_2 x$, with $\beta_1, \beta_2 \in \mathbb{F}_{q^{3m}}$, let $(1 + \beta_1)/\beta_2 = u_1 + u_2 y + u_3(y^2 - 2)$, then $\rho(\beta) = (u_2/u_1, u_3/u_1)$.

9.5.2 Decomposition Base

In this case the decomposition base consists of $\psi(at, 0)$, where a runs through all elements of $\mathbb{F}_q$ and t generates the polynomial basis, i.e.,

$$\mathcal{F} = \left\{ \frac{1 + (at)y + (1 - (at)^2)x}{1 + (at)y + (1 - (at)^2)x^2} : a \in \mathbb{F}_q \right\}$$

which clearly contains q elements, for much the same reason as given in Section 9.4. The reason for considering $\psi(at, 0)$ instead of $\psi(a, 0)$ is that the minimal polynomials of x and y are defined over $\mathbb{F}_q$. Note that this implies that $\psi(a, 0) \in T_6(\mathbb{F}_q)$ for $a \in \mathbb{F}_q$ and so does not generate a fixed proportion of $T_6(\mathbb{F}_{q^m})$, as is needed.

9.5.3 Relation Finding

Since $(\mathrm{Res}_{\mathbb{F}_{q^m}/\mathbb{F}_q} T_6)(\mathbb{F}_q)$ is $2m$-dimensional, we need to solve

$$\prod_{i=1}^{2m} P_i = R\,, \tag{9.7}$$

with $P_1, \ldots, P_{2m} \in \mathcal{F}$. Assuming that R is expressed in its canonical form, i.e., $R = \psi(r_1, r_2)$, we get

$$\prod_{i=1}^{2m} \left(\frac{1 + (a_i t)y + (1 - (a_i t)^2)x}{1 + (a_i t)y + (1 - (a_i t)^2)x^2} \right) = \frac{1 + r_1 y + r_2(y^2 - 2) + (1 - r_1^2 - r_2^2 + r_1 r_2)x}{1 + r_1 y + r_2(y^2 - 2) + (1 - r_1^2 - r_2^2 + r_1 r_2)x^2}.$$

After expanding the product of the numerators and denominators, the left hand side becomes the fairly general expression

$$\frac{b_0 + b_1 y + b_2(y^2 - 2) + (c_0 + c_1 y + c_2(y^2 - 2))\, x}{b_0 + b_1 y + b_2(y^2 - 2) + (c_0 + c_1 y + c_2(y^2 - 2))\, x^2} \tag{9.8}$$

with b_i, c_i polynomials over $\mathbb{F}_{q^m}$ of degree $4m$ in $a_1, \ldots, a_{2m}$. In general, these polynomials are rather large and thus difficult to work with.

Example 9.1. *For $m = 5$, the number of terms in the b_i (resp. c_i) is given by $B = [35956, 30988, 25073]$ (resp. $C = [35946, 31034, 24944]$) for finite fields of large characteristic.*

However, note that these polynomials are by construction symmetric in the $a_1, \ldots, a_{2m}$ so we can rewrite the b_i and c_i in terms of the $2m$ elementary symmetric polynomials $\sigma_j(a_1, \ldots, a_{2m})$ for $j = 1, \ldots, 2m$. This has quite a dramatic effect on the complexity of these polynomials, i.e., the degree is now only quadratic and the number of terms is much lower, since the maximum number of terms in a quadratic polynomial in $2m$ variables is $4m + \binom{2m}{2} + 1$.

Example 9.2. *For $m = 5$, when we consider the symmetric functions in the $a_i t$ to be the variables instead of simply a_i, the polynomials b_i and c_i are over the* *integers:*

$$\begin{aligned}
b_0 &= \sigma_2^2 + \sigma_2\sigma_5 - \sigma_2\sigma_8 - \sigma_3^2 - \sigma_3\sigma_6 + \sigma_3\sigma_9 - \sigma_3 + \sigma_5^2 + \sigma_5\sigma_8 - \sigma_6^2 - \sigma_6\sigma_9 \\
&\quad + \sigma_6 + \sigma_8^2 - \sigma_9^2 + 2\sigma_9 - 1 \\
b_1 &= \sigma_1\sigma_2 + \sigma_1\sigma_5 + \sigma_1\sigma_6 + \sigma_1\sigma_9 - \sigma_1 - \sigma_2\sigma_7 - \sigma_2\sigma_{10} - \sigma_3\sigma_4 - \sigma_3\sigma_7 + \sigma_4\sigma_5 \\
&\quad + \sigma_4\sigma_8 + \sigma_4\sigma_9 - \sigma_4 - \sigma_5\sigma_{10} - \sigma_6\sigma_7 - \sigma_6\sigma_{10} + \sigma_7\sigma_8 - \sigma_9\sigma_{10} + \sigma_{10} \\
b_2 &= \sigma_1\sigma_3 + \sigma_1\sigma_5 + \sigma_1\sigma_6 + \sigma_1\sigma_8 - \sigma_2\sigma_4 - \sigma_2\sigma_7 - \sigma_3\sigma_7 - \sigma_3\sigma_{10} + \sigma_4\sigma_6 \\
&\quad + \sigma_4\sigma_8 + \sigma_4\sigma_9 - \sigma_4 - \sigma_5\sigma_7 - \sigma_5\sigma_{10} - \sigma_6\sigma_{10} + \sigma_7\sigma_9 - \sigma_7 - \sigma_8\sigma_{10} \\
c_0 &= \sigma_1^2 + \sigma_1\sigma_4 - \sigma_1\sigma_7 - 2\sigma_1\sigma_{10} - \sigma_3^2 - \sigma_3\sigma_6 + \sigma_3\sigma_9 - \sigma_3 + \sigma_4^2 + \sigma_4\sigma_7 \\
&\quad - \sigma_4\sigma_{10} - \sigma_6^2 - \sigma_6\sigma_9 + \sigma_6 + \sigma_7^2 + \sigma_7\sigma_{10} - \sigma_9^2 + 2\sigma_9 + \sigma_{10}^2 - 1 \\
c_1 &= \sigma_1\sigma_2 + \sigma_1\sigma_5 - \sigma_2\sigma_3 - \sigma_2\sigma_6 - \sigma_2\sigma_7 - \sigma_2\sigma_{10} + \sigma_3\sigma_8 + \sigma_4\sigma_5 \\
&\quad + \sigma_4\sigma_8 - \sigma_5\sigma_6 - \sigma_5\sigma_9 - \sigma_5\sigma_{10} + \sigma_5 + \sigma_7\sigma_8 - \sigma_8\sigma_9 + \sigma_8 \\
c_2 &= \sigma_1\sigma_5 + \sigma_1\sigma_8 - \sigma_2\sigma_4 - \sigma_2\sigma_6 - \sigma_2\sigma_7 - \sigma_2\sigma_9 + \sigma_2 + \sigma_3\sigma_5 + \sigma_3\sigma_8 \\
&\quad + \sigma_4\sigma_8 - \sigma_5\sigma_7 - \sigma_5\sigma_9 - \sigma_5\sigma_{10} + \sigma_5 + \sigma_6\sigma_8 - \sigma_8\sigma_{10}
\end{aligned}$$

and the number of terms reduces to $B = [16, 19, 18]$ *and* $C = [20, 16, 16]$.

Note that the polynomials b_i and c_i only have to be computed once and can be reused for each random point R.

To generate the system of non-linear equations, we can proceed in two ways. The first method rewrites (9.8) in the canonical form (9.6) and then equates the two corresponding coefficients. Instead of using the inverse map ρ symbolically, we proceed as follows: multiplying both numerator and denominator of (9.8) by the two conjugates of $c_0+c_1y+c_2(y^2-2)$ and taking into account that $y^q = y^2-2$ and $(y^2-2)^q = -(y^2-2) - y$, we get

$$\frac{d_0 + d_1 y + d_2(y^2-2) + \Delta\, x}{d_0 + d_1 y + d_2(y^2-2) + \Delta\, x^2}\,, \tag{9.9}$$

with

$$\begin{aligned}
d_0 &= b_0c_0^2 - b_0c_1^2 + b_0c_1c_2 - b_0c_2^2 - 2b_1c_0c_1 + b_1c_0c_2 - b_1c_1^2 + 4b_1c_1c_2 - b_1c_2^2 \\
&\quad + b_2c_0c_1 - 2b_2c_0c_2 + 2b_2c_1^2 - 2b_2c_1c_2 - b_2c_2^2\,, \\
d_1 &= \; - b_0c_0c_1 + 2b_0c_1c_2 - b_0c_2^2 + b_1c_0^2 - b_1c_0c_2 + b_1c_1c_2 - 2b_1c_2^2 - b_2c_0c_1 \\
&\quad + b_2c_0c_2 - b_2c_1^2 + 2b_2c_1c_2\,, \\
d_2 &= \; - b_0c_0c_2 + b_0c_1^2 - b_0c_2^2 - b_1c_0c_1 + 2b_1c_1c_2 - b_1c_2^2 + b_2c_0^2 + b_2c_0c_2 \\
&\quad - 2b_2c_1^2 + b_2c_1c_2\,, \\
\Delta &= c_0^3 - 3c_0c_1^2 + 3c_0c_1c_2 - 3c_0c_2^2 - c_1^3 + 6c_1^2c_2 - 3c_1c_2^2 - c_2^3\,.
\end{aligned} \tag{9.10}$$

The system of $2m$ non-linear equations in the $2m$ unknowns $a_1, \ldots, a_{2m}$ is then given by the Weil restriction of the two equations

$$d_1 = r_1 d_0 \quad \text{and} \quad d_2 = r_2 d_0\,.$$

Note that d_0, d_1, d_2 are independent of R and thus only need to be computed once. The downside of this approach is that by computing the d_i explicitly via (9.10), the degrees and therefore also the numbers of terms explode. This makes it virtually impossible to use Gröbner basis techniques [20] to find the solutions of this system of non-linear equations. Hence we use the following method.

Instead, to solve the system of non-linear equations, we use the embedding of $T_6(\mathbb{F}_{q^m})$ into $T_2(\mathbb{F}_{q^{3m}})$ and consider the Weil restriction of the following equality:

$$\frac{b_0 + b_1y + b_2(y^2-2)}{c_0 + c_1y + c_2(y^2-2)} = \frac{1 + r_1y + r_2(y^2-2)}{1 - r_1^2 - r_2^2 + r_1r_2}\,.$$

The above equation leads to 3 non-linear equations over $\mathbb{F}_{q^m}$ or equivalently, to $3m$ non-linear equations over $\mathbb{F}_q$ in the $2m$ unknowns $\sigma_1, \ldots, \sigma_{2m}$. Note that amongst the $3m$ equations, there will be at least m dependent equations, caused by the fact that we only considered the embedding in T_2 and not strictly in T_6.

The efficiency with which one can find the solutions of this system of non-linear equations depends on many factors such as the multiplicities of the zeros or the number of solutions at infinity. For each random R, the resulting system of equations has the same structure, since only the value of some coefficients changes, but for finite fields of large enough characteristic, neither the degrees nor the numbers of terms. To determine the properties of these systems of equations we computed the Gröbner basis w.r.t. the lexicographic ordering using the Magma implementation of the F4-algorithm [35] and concluded the following:

- The ideal generated by the system non-linear equations is zero-dimensional, which implies that there is only a finite number of candidates for the σ_i.
- After homogenizing the system of equations, we concluded that there is only a finite number of solutions at infinity. This property is quite important, since we can then use an algorithm by Lazard [84] with proven complexity.
- The Gröbner basis w.r.t. the lexicographic ordering satisfies the so called Shape Lemma, i.e., the basis has the following structure:

  $$\sigma_1 - g_1(\sigma_{2m}),\ \sigma_2 - g_2(\sigma_{2m}),\ \ldots,\ \sigma_{2m-1} - g_{2m-1}(\sigma_{2m}),\ g_{2m}(\sigma_{2m})\,,$$

  where $g_i(\sigma_{2m})$ is a univariate polynomial in σ_{2m} for each i. By reducing modulo g_{2m} we can assume that $\deg(g_i) < \deg(g_{2m})$ and by Bezout's theorem we have $\deg(g_{2m}) \leq 2^{2m}$, since the non-linear equations are quadratic. However, our experiments show that in all cases we have $\deg(g_{2m}) = 3^m$.
- The polynomial $g_{2m}(\sigma_{2m})$ is squarefree, which implies that the ideal is in fact a radical ideal.

To test if a random point decomposes over the factor base, we first find the roots of $g_{2m}(\sigma_{2m})$ in $\mathbb{F}_q$, and then substitute these in the g_i to find the values of the σ_i for $i = 1, \ldots, 2m-1$. If there is more than one solution for the σ_i, then each is tested to see if it actually gives a relation, as follows. For each such $2m$-tuple, we then test if the polynomial

$$p(x) := x^{2m} - \sigma_1 x^{2m-1} + \sigma_2 x^{2m-2} - \cdots + (-1)^{2m}\sigma_{2m}$$

splits completely over $\mathbb{F}_q$. If it does, then the roots a_i for $i = 1, \ldots, 2m$ lead to a possible relation of the form (9.7).

9.5.4 Complexity Analysis and Experiments

The probability of obtaining a relation is now $1/(2m)!$ and since the factor base again consists of q elements, we need to perform on average $(2m)!q$ decompositions. Each decomposition consists of the following steps:

- Since the polynomials b_i and c_i only need to be computed once, generating the system of non-linear equations requires $O(1)$ multiplications of multivariate polynomials with $O(m^2)$ terms with an $\mathbb{F}_{q^m}$-element. Using a naive multiplication routine, the overall time to generate one such system is therefore $O(m^4)$ operations in $\mathbb{F}_q$.

- Computing the Gröbner basis using the F5-algorithm algorithm [36] requires $O(\binom{4m}{2m}^\omega)$ operations in $\mathbb{F}_q$, with ω the complexity of matrix multiplication, i.e., $\omega = 3$ using a naive algorithm. Using the fact that

$$\binom{2n}{n} \cong \sqrt{\frac{\pi}{2}}(2n)^{-1/2}2^{2n} = O(2^{2n})$$

  we obtain a complexity of $O(2^{12m})$ operations in $\mathbb{F}_q$.

- Since $\deg(g_{2m}) = 3^m$, computing $\gcd(g_{2m}(z), z^q - z)$ requires $O(3^{2m} \log q)$ operations in $\mathbb{F}_q$. On average, the polynomial will have one root in $\mathbb{F}_q$, so finding the actual roots takes negligible time.

- Testing if the polynomial $p(x)$ has roots in $\mathbb{F}_q$ requires $O(m^2 \log q)$ operations in $\mathbb{F}_q$. Since this only happens with probability $1/(2m)!$, when it does split, finding the actual roots is negligible.

The overall time complexity to generate sufficient relations therefore amounts to

$$O\left((2m)! \cdot q \cdot (2^{12m} + 3^{2m} \log q)\right)$$

operations in $\mathbb{F}_q$.

Finding an element in the kernel of a matrix of dimension q with $2m$ non-zero elements per row requires $O(mq^2)$ operations in $\mathbb{Z}/(\Phi_6(q^m)\mathbb{Z})$, which finally justifies the following complexity estimate:

Run Time Heuristic 9.1. *The expected running time of the T_6-algorithm to compute discrete logarithms in $T_6(\mathbb{F}_{q^m})$ is*

$$O((2m)! \cdot q \cdot (2^{12m} + 3^{2m} \log q) + m^3 q^2)$$

operations in $\mathbb{F}_q$.

Again, the results of [49, 106, 154] imply that the complexity can be reduced to $O(q^{2-1/m})$ as $q \to \infty$, since in this case the dimension is $2m$.

The expected running time of the T_6-algorithm is minimal precisely when the relation collection stage takes about the same time as the linear algebra stage, i.e., when $(2m)! \cdot 2^{12m} \simeq q$. Note that for such q and m, the term $3^{2m} \log q$ is negligible compared to 2^{12m}. The overall running time then again becomes

$$\begin{aligned} O(m^3 q^2) &= O\big(\exp(3 \log m + 2 \log q)\big) \\ &= O\big(\exp(5 (\log q)^{1/2} (\log q)^{1/2})\big) \\ &= O\big(\exp(5 (2m \log 2m + 12m)^{1/2} (\log q)^{1/2})\big) \\ &= O\big(L_{q^m}(1/2, c')\big) \end{aligned}$$

with $c' \in \mathbb{R}_{>0}$. Note that for the first equality we have used again $\log m < \log q$, and for the second and third equality we have used $\log q \simeq 2m \log m + 12m \log 2$.

The practicality of the T_6-algorithm clearly depends on the efficiency of the Gröbner basis computation. Note that for small m, the complexity of the Gröbner basis computation is greatly overestimated by the $O(2^{12m})$ operations in $\mathbb{F}_q$.

Due to the use of the symmetric polynomials, the input polynomials are only quadratic instead of degree $4m$. As one can see from Figure 9.2, this makes the algorithm quite practical. The table should be interpreted as for Figure 9.1, i.e., the torus size is constant across each row and for a given size q^m, the table contains

Figure 9.2: $\log_2$ of expected running times (s) of the T_6-algorithm and Pollard-Rho in a subgroup of size 2^{160}: bold for time $< 2^{45}$ and matrix of size $< 2^{23}$.

			m				
$\log_2 \lvert\mathbb{F}_{p^{6m}}\rvert$	$\log_2 \lvert T_6(\mathbb{F}_{p^m})\rvert$	ρ	1	2	3	4	5
200	67	18	25	**18**	**14**	**20**	**29**
300	100	34	42	36	**21**	**24**	**32**
400	134	52	59	54	**32**	**29**	**36**
500	167	66	75	71	44	**33**	**39**
600	200	66	93	88	55	40	**42**
700	234	66	109	105	67	48	46
800	267	66	127	122	78	57	51
900	300	68	144	139	90	65	56
1000	334	69	161	156	101	74	60

for $m = 1, \dots, 5$, the $\log_2$ of the expected running times in seconds for the entire algorithm. Taking into account the memory restrictions on the matrix, i.e., the dimension should be limited by 2^{23}, the timings given in bold are feasible with the current Magma implementation.

Remark 9.2. *Note that the column for $m = 5$ provides an upper bound for the hardness of the DLP in $T_{30}(\mathbb{F}_q)$, since this can be embedded in $T_6(\mathbb{F}_{q^5})$. This group was proposed in Chapter 5 and also in [85] for cryptographic use where keys of length 960 bits were recommended, i.e., with q of length 32 bits. Figure 9.2 shows that even with a Magma implementation it would be feasible to compute discrete logarithms in $T_{30}(\mathbb{F}_p)$ with p a prime of around 20 bits. The embedding in $T_2(\mathbb{F}_{p^{15}})$ is about 2^{10} times less efficient as can be seen from the column for $m = 15$ in Figure 9.1. In light of this attack, the security offered by the DLP in finite fields of the form $\mathbb{F}_{q^{30}}$ should be completely reassessed. Note that by simply comparing the complexities given in Theorem 1 and the above run time heuristic, it is a priori not clear that the T_6-algorithm is in fact faster than the corresponding T_2-algorithm. This phenomenon is caused by the overestimating the complexity of the Gröbner basis computation.*

9.5.5 Comparison with other Methods

In this section we compare the T_6-algorithm with the Pollard-Rho and the Adleman-DeMarrais index calculus algorithm.

Pollard-Rho in the Full Torus

Since the size of $T_6(\mathbb{F}_{q^m})$ is given by $\Phi_6(q^m) \simeq q^{2m}$, we conclude that the Pollard-Rho algorithm takes, in the worst case, $O(q^m)$ operations in $T_6(\mathbb{F}_{q^m})$ or $O(m^2 q^m)$ operations in $\mathbb{F}_q$. If we assume that q is large enough such that the term q^2 determines the overall running time, i.e., $(2m)!2^{12m} \leq q$, then the T_6-algorithm will be at least as fast as Pollard-Rho whenever $m \geq 3$. Again we note that the T_6 algorithm does not lead to an improvement over the existing attacks on LUC [146], XTR [88], CEILIDH [127] or MNT curves [104] as long as these systems are defined over $\mathbb{F}_p$. However, the security of XTR over extension fields, as proposed in [93] and the proposals of Chapter 5, needs to be reassessed as shown below.

Pollard-Rho in a Subgroup of prime order $\simeq 2^{160}$

As for the T_2-algorithm, the third column of Figure 9.2 contains the expected running time of the Pollard-Rho algorithm in a subgroup of $T_6(\mathbb{F}_{q^m})$ of prime order l with $l \simeq 2^{160}$. In this case, the column for $m = 5$ gives an upper bound of the security of the T_{30} cryptosystem of Chapter 5. As is clear from Figure 9.2, for $m = 5$, our algorithm is always faster than Pollard-Rho, and the matrices occurring in the linear algebra step would be feasible up to 700-bit fields.

However, as was the case for the T_2-algorithm, the importance of Figure 9.2 is that it contains the first practical upper bounds for the hardness of the DLP in extension fields $\mathbb{F}_{q^{6m}}^{\times}$.

9.6 Conclusion and Future Work

In this chapter we have presented an index calculus algorithm, following ideas of Gaudry, to compute discrete logarithms on rational algebraic tori. Our algorithm works directly in the torus and depends fundamentally on the compression mechanisms previously used in a constructive context for systems such as LUC, XTR and CEILIDH.

We have also provided upper bounds for the difficulty of solving discrete logarithms on the tori $T_2(\mathbb{F}_{q^m})$ and $T_6(\mathbb{F}_{q^m})$ for various q and m in the cryptographic range. These upper bounds indicate that if the techniques in this chapter can be made fully practical and optimized, then they may weaken the security of practical systems based on T_{30}, for fields in the 1000 bit range.

In the near future we wish to investigate the approach by Diem [27], who allows a larger decomposition base when necessary. The disadvantage of this approach is that it destroys the symmetric nature of the polynomials defining the decomposition of a random element over the factor base, which makes Gröbner basis techniques virtually impossible.

It is clear that the Magma implementations described in this chapter are not optimised and many possible improvements exist. Two factors mainly determine the running time of the algorithm: first of all, the probability that a random element decomposes over the factor base and secondly, the time it takes to solve a system of non-linear equations over a finite field. The first factor could be influenced by designing some form of sieving, if at all possible, whereas the second factor could be improved by exploiting the fact that many very similar Gröbner bases have to be computed.

One possibility for overcoming the former problem would be to eliminate this probability entirely, as follows. Using the T_2-algorithm for this example, instead of having just one factor base $\mathcal{F}$ as in (9.4), we use m factor bases $\mathcal{F}_1, \ldots, \mathcal{F}_m$, where

$$\mathcal{F}_i = \left\{ \frac{at^{i-1} - \gamma}{at^{i-1} + \gamma} : a \in \mathbb{F}_q \right\} \subset T_2(\mathbb{F}_{q^m}).$$

One then expects that *every* point $R \in T_2(\mathbb{F}_{q^m})$ admits a unique decomposition $R = P_1 + \cdots + P_m$, with $P_i \in \mathcal{F}_i$, since the set of all sums of m elements from the $\mathcal{F}_i$ has cardinality $q^m \approx \#T_2(\mathbb{F}_{q^m})$. The drawback of this method is that since the system is no longer symmetric, the decomposition method no doubt becomes far more costly.

Vercauteren and Lercier [159], using the approach described in this chapter, were able to compute discrete logarithms in a field $\mathbb{F}_{p^{18}}$ of size 336 bits in less than one week. At the time the research contained in this chapter was carried out, the index calculus algorithm of Adleman and DeMarrais [2] was the best known subexponential algorithm for fields of medium extension degree, i.e., those with degrees from ten to thirty or so.

However, recently Joux and Lercier [67, 68] extended Adleman's FFS method to fields of this type, resulting in an $L[1/3, c]$ algorithm [**?**] for some combinations of p and n, thus beating the complexity of the Adleman-DeMarrais algorithm. Impressively, they were able to compute discrete logarithms in a field $\mathbb{F}_{p^{30}}$ of size 556 bits, in less than twelve hours on a 1.15 GHz 16-processors HP AlphaServer GS1280. The main difference between their algorithm and the one presented here is that the relation generation is far more efficient, while the linear algebra stage for both is $O(p^2)$. It therefore seems that the approach of Joux and Lercier, as well as being more generally applicable than the algorithm described in this chapter, is more efficient in practice as well.

More recently, using several new ideas for the Number Field Sieve, Joux, Lercier, Vercauteren and Smart [71] described a family of algorithms that have $L[1/3, c]$ complexity, for all remaining combinations of p and n that had until then eluded an $L[1/3]$ complexity. Thus there now exists an $L[1/3, c]$ for *all* finite fields, with various constants, as has been believed by several experts for some time.

Final Remarks

In this thesis we have presented several techniques for improving both the time and space efficiency of cryptographic systems that are based in, or map to, the multiplicative group of finite fields with small extension degree. We have also presented a security analysis of the systems proposed, using both existing algorithms, and developing our own, revealing weaknesses in current torus-based cryptographic parameters that were not previously known.

The results of the thesis raise two obvious questions. While the torus interpretation of subgroups of $\mathbb{F}_{q^n}^{\times}$ allows for asymptotically optimal element representation, do there exist any other algebraic interpretations that might allow faster arithmetic? Secondly, while the algorithm presented in Chapter 9 demonstrates a weakness in some medium degree extension fields, the methods employed are in some sense artificial, since they rely on an imposed algebraic notion of smoothness. Therefore, might there exist algorithms which exploit the torus arithmetic more directly?

Naturally any method that offers a solution to the former problem has possible implications for the latter, in that any algebraic interpretation may give rise to an attack if there exists a corresponding DLP algorithm for the interpretation. In fact a preprint by Kohel [77] suggests a possible method along these lines, by modelling algebraic tori as subschemes of the generalised Jacobian of singular hyperelliptic curves. Unfortunately the arithmetic obtained is less efficient than that detailed in Part I of this thesis. Furthermore, it is not difficult to see that the proposed method really only models T_2, with the higher-dimensional tori arising simply as subschemes. Hence an attack on the DLP via this approach will only provide the same benefits as the T_2 algorithm presented in Chapter 9. Indeed it was the failure of this approach to give a compact representation of the higher tori which led to the idea of using the rationality of T_6 to gain a tighter representation, and hence a higher probabilty of relation generation using Gaudry's method [47].

The fact that such models exist however implies that it may be possible to efficiently represent, and hence attack, systems based on higher-dimensional algebraic tori, and hence extension fields, using more sophisticated ideas than are presently known.

Bibliography

[1] L. M. Adleman. *The function field sieve*. In Algorithmic Number Theory Symposium (ANTS-I), Springer LNCS 877, 108–121, 1994.

[2] L. M. Adleman and J. DeMarrais. *A subexponential algorithm for discrete logarithms over all finite fields*. Mathematics of Computation, **61 (203)**, 1–15, 1993.

[3] L. M. Adleman and M. A. Huang. *Function field sieve method for discrete logarithms over finite fields*. Journal of Information and Computation, **151 (1-2)**, 5–16, 1999.

[4] R. M. Avanzi. *Aspects of Hyperelliptic Curves over Large Prime Fields in Software Implementations.* In Cryptographic Hardware and Embedded Systems (CHES 2004), Springer LNCS 3156, 148–162, 2004.

[5] P. Barreto. Personal Communication.

[6] P. Barreto, S. Galbraith, C. Ó hÉigeartaigh, and M. Scott. *Efficient Pairing Computation on Supersingular Abelian Varieties.* Cryptology ePrint Archive, Report 2004/375. Available from `http://eprint.iacr.org/2004/375`.

[7] P. Barreto, H. Kim, B. Lynn and M. Scott. *Efficient Algorithms for Pairing-Based Cryptosystems.* In Advances in Cryptology (CRYPTO 2002), Springer LNCS 2442, 354–368, 2002.

[8] E. A. Bender and E. R. Canfield. *An approximate probabilistic model for structured Gaussian elimination.* Journal of Algorithms, **31**, 271–290, 1999.

[9] G. Bertoni, J. Guajardo, S. Kumar, G. Orlando, C. Paar and T. Wollinger. *Efficient GF(p^m) arithmetic architectures for cryptographic applications.* In Topics in Cryptology – CT-RSA, Springer LNCS 2612, 158–175, 2003.

[10] I. F. Blake, G. Seroussi and N. P. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, 1999.

[11] I. F. Blake, G. Seroussi and N. P. Smart. *Advances in Elliptic Curve Cryptography.* Cambridge University Press, 2005.

[12] D. Boneh and X. Boyen. *Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles.* In Advances in Cryptology (EUROCRYPT 2004), Springer LNCS 3027, 223–238, 2004.

[13] D. Boneh, X. Boyen and H. Shacham. *Short Group Signatures.* In Advances in Cryptology (CRYPTO 2004), Springer LNCS 3152, 41–55, 2004.

[14] D. Boneh and M. Franklin. *Identity-Based Encryption from the Weil Pairing.* SIAM Jounal on Computing, Volume 32, no. 3, 586-615, 2003.

[15] D. Boneh, B. Lynn and H. Shacham. *Short signatures from the Weil Pairing.* In Advances in Cryptology (ASIACRYPT 2001), Springer LNCS 2248, 514–532, 2001.

[16] D. Boneh and R. Venkatesan. *Breaking RSA may not be equivalent to factoring (extended abstract).* In Advances in Cryptology (EUROCRYPT 1998), Springer LNCS 1403, 59–71, 1998.

[17] W. Bosma, J. Hutton and E. Verheul. *Looking beyond XTR.* In Advances in Cryptology (ASIACRYPT 2002), Springer LNCS 2501, 46–63, 2002.

[18] A. Bosselaers, R. Govaerts and J. Vandewalle. *Comparison of Three Modular Reduction Functions.* In Advances in Cryptology (CRYPTO 1994) Springer LNCS 773, 175–186, 1994.

[19] A. E. Brouwer, R. Pellikaan and E. R. Verheul. *Doing more with fewer bits*. In Advances in Cryptology (ASIACRYPT 1999), Springer LNCS 1716, 321–332, 1999.

[20] B. Buchberger. *A theoretical basis for the reduction of polynomials to canonical forms*. ACM SIGSAM Bull., **10 (3)**, 19–29, 1976.

[21] W. Clark and J. Liang. *On arithmetic weight for a general radix representation of integers.* IEEE Trans. Info. Theory, **19**, 823–826, 1973.

[22] C. C. Cocks. *A Note on Non-Secret Encryption (1973).* Available from: http://www.cesg.gov.uk/site/publications/media/notense.pdf

[23] H. Cohen, A. Miyaji and T. Ono. *Efficient elliptic curve exponentiation using mixed coordinates.* In Advances in Cryptology (ASIACRYPT 1998). Springer LNCS 1514 , 51–65, 1998.

[24] D. Coppersmith. *Evaluating logarithms in GF(2^n).* In 16th ACM Symp. Theory of Computing, 201–107, 1984.

[25] D. Coppersmith. *Fast evaluation of logarithms in fields of characteristic two.* IEEE Trans. Info. Theory, **30**, 587–594, July 1984.

[26] J. Daemen and V. Rijmen. *The design of Rijndael: AES - the Advanced Encryption Standard.* Springer, 2002.

[27] C. Diem. *On the discrete logarithm problem in elliptic curves over non-prime fields*. Talk at ECC 2004. Available from the author.

[28] W. Diffie and M. E. Hellman. *New directions in cryptography*. IEEE Trans. Inform. Theory **22 (6)**, 644–654, 1976.

[29] R. Dutta, R. Barua and P. Sarkar. *Pairing-Based Cryptographic Protocols: A Survey.* Cryptology ePrint Archive, Report 2004/064. Available from `http://eprint.iacr.org/2004/064`.

[30] I. Duursma and H. Lee. *Tate Pairing Implementation for Hyperelliptic Curves* $y^2 = x^p - x + d$. In Advances in Cryptology (ASIACRYPT 2003), Springer LNCS 2894, 111–123, 2003.

[31] A. K. Ekert. *Quantum cryptography based on Bell's theorem.* Phys. Rev. Lett. 67, 661–663 (1991).

[32] T. ElGamal. *A public key cryptosystem and a signature scheme based on discrete logarithms.* In Advances in Cryptology (CRYPTO 1984), Springer LNCS 196, 10–18, 1985.

[33] J. H. Ellis. *The Possibility of Non-Secret Encryption (1970).* Available from: http://www.cesg.gov.uk/site/publications/media/possnse.pdf

[34] A. Enge and P. Gaudry. *A general framework for subexponential discrete logarithm algorithms.* Acta Arithmetica, 102, 83-103 2002.

[35] J.-C. Faugère. *A new efficient algorithm for computing Gröbner bases* (F_4), J. Pure Appl. Algebra **139 (1-3)**, 61-88, 1999.

[36] J.-C. Faugère. *A new efficient algorithm for computing Gröbner bases without reduction to zero* (F_5), In Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, 75–83, 2002.

[37] FIPS 46. *Data Encryption Standard.* Federal Information Processing Standard (FIPS), National Bureau of Standards, U.S. Department of Commerce, Washington D.C., 1977.

[38] FIPS 46-2. *Data Encryption Standard.* Federal Information Processing Standard (FIPS), National Bureau of Standards, U.S. Department of Commerce, Washington D.C., 1993.

[39] FIPS 186-2. *Digital Signature Standard.* Federal Information Processing Standards Publication 186-2, 2000.

[40] FIPS 197. *Advanced Encryption standard*. Federal Information Processing Standards (FIPS), National Institute of Standards and Technology, U.S. Department of Commerce, Washington D.C., 2001.

[41] G. Frey and H. Ruck. *A Remark Concerning m-Divisibility and the Discrete Logarithm Problem in the Divisor Class Group of Curves.* Mathematics of Computation, **62**, 865-874, 1994.

[42] S. Galbraith. *Supersingular Curves in Cryptography.* In Advances in Cryptology (ASIACRYPT 2001), Springer LNCS 2248, 495–513, 2001.

[43] S. Galbraith, K. Harrison and D. Soldera. *Implementing the Tate pairing.* In Algorithmic Number Theory Symposium (ANTS-IV) Springer LNCS 2369, 324–337, 2002.

[44] R. Gallant, J. Lambert and S. Vanstone. *Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms.* In Advances in Cryptology (CRYPTO 2001), Springer LNCS 2139, 190–200, 2001.

[45] S. Gao. *Normal Bases over Finite Fields.* PhD Thesis, University of Waterloo, 1993.

[46] S. Gao and J. Howell. *A general polynomial sieve.* Designs, Codes and Crpyotgraphy, vol. 18, 149–157, 1999.

[47] P. Gaudry *Index calculus for abelian varieties and the elliptic curve discrete logarithm problem.* Cryptology ePrint Archive, Report 2004/073. Available from `http://eprint.iacr.org/2004/073`.

[48] P. Gaudry, F. Hess and N. P. Smart. *Constructive And Destructive Facets Of Weil Descent On Elliptic Curves.* J. of Cryptology, **15**(1), 19–46, 2002.

[49] P. Gaudry, E. Thomé, N. Thériault and C. Diem. *A double large prime variation for small genus hyperelliptic index calculus*. Cryptology ePrint Archive, Report 2004/153. Available from `http://eprint.iacr.org/2004/153`.

[50] C. Gentry. *Certificate-Based Encryption and the Certificate Revocation Problem.* In Advances in Cryptology (EUROCRYPT 2003), Springer LNCS 2656, 272–293, 2003.

[51] P. Golle and A. Juels. *Dining Cryptographers Revisited.* In Advances in Cryptology (EUROCRYPT 2004), Springer LNCS 3027, 456–473, 2004.

[52] D. M. Gordon. *Discrete Logarithms in $GF(p)$ Using the Number Field Sieve.* SIAM J. of Disc. Math., **6**, 124–138, 1993.

[53] D. M. Gordon and K. S. McCurley. *Massively parallel computation of discrete logarithms.* In Advances in Cryptology – CRYPTO 1992, Springer LNCS 740, 312–323, 1993.

[54] R. Granger. *Estimates for discrete logarithm computations in finite fields of small characteristic.* In Cryptography and Coding, Springer LNCS 2898, 190–206, 2003.

[55] R. Granger, A. Holt, D. Page, N. P. Smart and F. Vercauteren. *Function Field Sieve in Characteristic Three.* In Algorithmic Number Theory Symposium (ANTS-VI), Springer LNCS 3076, 223–234, 2004.

[56] R. Granger, D. Page and M. Stam. *A Comparison of CEILIDH and XTR.* In Algorithmic Number Theory Symposium (ANTS-VI) Springer LNCS 3076, 235–249, 2004.

[57] R. Granger, D. Page and M. Stam. *On Small Characteristic Algebraic Tori in Pairing-based Cryptography.* In LMS Journal of Computation and Mathematics, Volume 9: 64–85, 2006.

[58] R. Granger, D. Page and M. Stam. *Hardware and Software Normal Basis Arithmetic for Pairing-Based Cryptography in Characteristic Three.* IEEE Transactions on Computers 54(7), 852–860, 2005.

[59] R. Granger and F. Vercauteren. *On the Discrete Logarithm Problem on Algebraic Tori.* In Advances in Cryptology (CRYPTO 2005), Springer LNCS 3621, 66–85, 2005.

[60] K. Harrison, D. Page and N. P. Smart. *Software Implementation of Finite Fields of Characteristic Three, for use in Pairing Based Cryptosystems.* LMS Journal of Computation and Mathematics, **5** (1), 181–193, London Mathematical Society, 2002.

[61] R. Hartshorne. *Algebraic Geometry,* Springer, 1997.

[62] F. Hess. *Efficient Identity based Signature Schemes based on Pairings.* In Selected Areas in Cryptography (SAC 2002), Springer LNCS 2595, 310–324, 2003.

[63] F. Hess, F. Vercauteren and N. Smart. *The Eta Pairing Revisited.* Preprint, March 2006.

[64] IEEE 1363, IEEE standard specifications for public key cryptography, 2000.

[65] T. Itoh and S. Tsujii. *A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases.* Info. and Comp., **78**(3), 171–177, 1988.

[66] A. Joux. *A One Round Protocol for Tripartite Diffie-Hellman.* In Algorithmic Number Theory Symposium (ANTS-IV), Springer LNCS 1838, 385–394, 2000.

[67] A. Joux and R. Lercier. *Discrete logarithms in $GF(65537^{25})$ – 120 digits – 400 bits*: email to NMBRTHRY list, October 2005.

[68] A. Joux and R. Lercier. *Discrete logarithms in $GF(370801^{30})$ – 168 digits – 556 bits*: email to NMBRTHRY list, October 2005.

[69] A. Joux and R. Lercier. *The Function Field Sieve in the Medium Prime Case.* To appear in Advances in Cryptology (EUROCRYPT 2006).

[70] A. Joux and R. Lercier. *The Function Field Sieve is Quite Special.* In Algorithmic Number Theory Symposium (ANTS-V), Springer LNCS 2369, 431–445, 2002.

[71] A. Joux, R. Lercier, F. Vercauteren and N. Smart. *The Number Field Sieve in the Medium Prime Case.* Preprint.

[72] D. E. Knuth. *The Art of Computer Programming. Vol. 2,* Addison-Wesley, 1981.

[73] N. Koblitz. *An elliptic curve implementation of the finite field digital signature algorithm.* In Advances in Cryptology (CRYPTO 1998), Springer LNCS 1462, 327–337, 1998.

[74] N. Koblitz. *Elliptic Curve Cryptosystems.* Mathematics of Computation, **48**, 203–209, 1987.

[75] N. Koblitz. *Hyperelliptic Cryptosystems.* In J. of Cryptology, **1**, 139–150, 1989.

[76] N. Koblitz and A. J. Menezes. *Pairing-Based Cryptography at High Security Levels.* Cryptology ePrint Archive, Report 2005/076. Available from `http://eprint.iacr.org/2005/076`.

[77] D. Kohel. *Constructive and destructive facets of torus-based cryptography.* Available from the author.

[78] M. Kraitchik. *Théorie des nombres.* Vol. 1, Gauthiers-Villars, 1922.

[79] M. Kraitchik. *Recherches sur la théorie des nombres.* Gauthiers-Villars, 1924.

[80] S. Kwon. *Efficient Tate Pairing Computation for Elliptic Curves over Binary Fields.* In Proc. of ACISP 2005, Springer LNCS 3574, 134–145, 2005.

[81] A. A. Klyachko. *On the Rationality of Tori with Cyclic Splitting Field (Russian).* Arithmetic and Geometry of Varieties, Kuybyshev Univ. Press, 73–78, 1988.

[82] B. A. LaMacchia and A. M. Odlyzko. *Solving large sparse linear systems over finite fields.* In Advances in Cryptology (CRYPTO 1990), Springer LNCS 537, 109–133, 1991.

[83] C. Lanczos. *Solution of systems of linear equations by minimized iterations.* J. of Research of the National Bureau of Standards, **49**, 33–53, 1952.

[84] D. Lazard. *Résolution des systèmes d'équations algébriques,* Theoret. Comput. Sci., **15 (1)**, 77–110, 1981.

[85] A. K. Lenstra. *Using Cyclotomic Polynomials to Construct Efficient Discrete Logarithm Cryptosystems over Finite Fields.* In Proc. of ACISP '97, Springer LNCS 1270, 127–138, 1997.

[86] A. K. Lenstra and H. W. Lenstra Jr. *The development of the number field sieve.* Springer LNM 1554, 1993.

[87] A. K. Lenstra and H. W. Lenstra Jr. *Algorithms in number theory.* Technical Report 87-008, University of Chicago, 1987.

[88] A. K. Lenstra and E. Verheul. *The XTR Public Key System.* In Advances in Cryptology (CRYPTO 2000), Springer LNCS 1880, 1–19, 2000.

[89] A. K. Lenstra and E. Verheul. *An Overview of the XTR Public Key System.* In Public Key Cryptography and Computational Number Theory, Verlages Walter de Gruyter, 151–180, 2001.

[90] H. W. Lenstra, Jr. *Factoring integers with elliptic curves.* Annals of Mathemetics (2), 126(3), 649–673, 1987.

[91] H. W. Lenstra Jr. *Finding Isomorphisms Between Finite Fields.* Mathematics of Computation, **56**, number 193, 329–347, 1991.

[92] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications, revised edition.* Cambridge, England: Cambridge University Press, 1994.

[93] S. Lim, S. Kim, I. Yie, J. Kim and H. Lee. *XTR extended to GF(p^{6m}).* In Selected Areas in Cryptography (SAC 2001), Springer LNCS 2259, 301–312, 2001.

[94] C. H. Lim and P. J. Lee. *More flexible exponentiation with precomputation.* In Advances in Cryptology (CRYPTO), Springer LNCS 839, 95–107, 1994.

[95] J. López and R. Dahab. *High Speed Software Multiplication in $\mathbb{F}_{2^m}$.* In Progress in Cryptography (INDOCRYPT 2000), Springer LNCS 1977, 203–212, 2000.

[96] R. Matsumoto. *Using C_{ab} Curves in the Function Field Sieve.* IEICE Trans. Fundamentals, **82**, March 1999.

[97] U. M. Maurer and S. Wolf. *The Diffie-Hellman Protocol.* Designs, Codes and Cryptography (19), 147–171, 2000.

[98] K. McCurley. *The discrete logarithm problem.* Cryptology and computational number theory, Proc. Symp. in Applied Mathematics 42, American Mathematical Society, pp. 49–74, 1990.

[99] G. C. Meletiou. *Explicit Form for the Discrete Logarithm over the Field GP(p, k).* Archivum Mathematicum (BRNO) 29, 25–28, 1993.

[100] A. J. Menezes, T. Okamoto and S. A. Vanstone. *Reducing elliptic curve logarithms to logarithms in a finite field.* IEEE Transactions in Information Theory, **39**, 1639–1646, 1993.

[101] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[102] V. Miller. *Short programs for functions on curves.* Unpublished manuscript, 1986. Available from `http://crypto.stanford.edu/miller/miller.pdf`.

[103] V. Miller. *Uses of Elliptic Curves in Cryptography.* In Advances in Cryptology (CRYPTO 1985), Springer LNCS 218, 417–426, 1985.

[104] A. Miyaji, M. Nakabayashi and S. Takano. *New explicit conditions of elliptic curve traces for FR-reduction*. IEICE Trans. Fundamentals **E84-A (5)**, 1234–1243, 2001.

[105] P. L. Montgomery. *Modular Multiplication Without Trial Division.* Mathematics of Computation, **44**, 519–521, 1985.

[106] K. Nagao. *Improvement of Thériault algorithm of index calculus for Jacobian of hyperelliptic curves of small genus*. Cryptology ePrint Archive, Report 2004/161. Available from `http://eprint.iacr.org/2004/161`.

[107] V. Nechaev. *Complexity of a determinate algorithm for the discrete logarithm.* In Mathematical Notes, 55(2), 165–172, 1994. Translated from Matematicheskie Zametki 55(2), 91–101, 1994.

[108] M. A. Neilsen and I. L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.

[109] K. Nguyen. *Explicit Arithmetic of Brauer Groups.* PhD Thesis, Universität Gesamthochschule Essen, 2001.

[110] P. Ning and Y. L. Yin. *Efficient Software Implementation for Finite Field Multiplication in Normal Basis.* In Information and Communications Security (ICICS), Springer LNCS 2229, 177–188, 2001.

[111] M. Nöcker. *Data structures for parallel exponentiation in finite fields.* PhD Thesis, Universität Paderborn, 2001.

[112] See: `http://www.nsa.gov/ia/industry/crypto_suite_b.cfm`

[113] A. M. Odlyzko. *Discrete logarithms in finite fields and their cryptographic significance*. In Advances in Cryptology (EUROCRYPT 1984), Springer LNCS 209, 224–314, 1985.

[114] A. M. Odlyzko. *Discrete logarithms and smooth polynomials.* Finite Fields: theory, applications, and algorithms, Contemp. Math 168, American Mathematical Society, pp. 269–278, 1994.

[115] T. Okamoto and R. Kashima. *Resource Bounded Unprovability of Computational Lower Bounds.* Cryptology ePrint Archive, Report 2003/187. Available from `http://eprint.iacr.org/2003/187`.

[116] D. Page. *Parallel Solution of Sparse Linear Systems Defined Over GF(p).* Technical Report, University of Bristol. November 2004.

[117] D. Page and N. P. Smart. *Hardware implementation of finite fields of characteristic three.* In Cryptographic Hardware and Embedded Systems (CHES 2002), Springer-Verlag LNCS 2523, 529–539, 2002.

[118] D. Page, N. P. Smart and F. Vercauteren. *A comparison of MNT curves and supersingular curves.* Cryptology ePrint Archive, Report 2004/165. Available from `http://eprint.iacr.org/2004/165`.

[119] G. Pohlig and M. Hellman. *An improved algorithm for computing discrete logarithms over $GF(p)$ and its cryptographic significance.* IEEE Trans. Info. Theory **24**, 106–110, 1978.

[120] J. Pollard. *Monte Carlo methods for index computation (mod p).* Math. Comp., **32(143)**, 918–924, 1978.

[121] C. Pomerance and J. W. Smith. *Reduction of huge, sparse linear systems over finite fields via created catastrophes.* Exper. Math., **1**, 89–94, 1992.

[122] J. Proos. *Joint Sparse Forms and Generating Zero Columns when Combing.* University of Waterloo, Technical Report CORR 2003-23,

[123] A. Reyhani-Masoleh and M. A. Hasan: *Fast Normal Basis Multiplication Using General Purpose Processors.* In Selected Areas in Cryptography (SAC 2001), Springer LNCS 2259, 230–244, 2001.

[124] R. L. Rivest, A. Shamir and L. Adleman. *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, 21(2):120–126, 1978.

[125] K. Rubin and A. Silverberg. *Algebraic Tori in Cryptography.* In High Primes and Misdemeanours: Lectures in Honour of the 60th birthday of Hugh Cowie Williams, Fields Institute Communications Series 41, American Mathematical Society, 317–326, 2004.

[126] K. Rubin and A. Silverberg. *Supersingular abelian varieties in cryptology.* In Advances in Cryptology (CRYPTO 2002), Springer LNCS 2442, 336–353, 2002.

[127] K. Rubin and A. Silverberg. *Torus-Based Cryptography.* In Advances in Cryptology (CRYPTO 2003), Springer LNCS 2729, 349–365, 2003.

[128] K. Rubin and A. Silverberg. *Using Primitive Subgroups to Do More with Fewer Bits.* In Algorithm Number Theory (ANTS-VI), Springer LNCS 3076, 18–41, 2004.

[129] R. Sakai, K. Ohgishi and M. Kasahara. *Cryptosystems Based on Pairings.* In Symposium on Cryptography and Information Security 2000 (SCIS2000), Okinawa, Japan, Jan 26–28, 2000.

[130] O. Schirokauer. *The number field sieve for integers of low weight.* Cryptology ePrint Archive, Report 2006/107. Available from `http://eprint.iacr.org/2006/107`.

[131] O. Schirokauer. *The special function field sieve.* SIAM Journal on Discrete Mathematics, **16**, 81–98, 2002.

[132] O. Schirokauer. *Using number fields to compute logarithms in finite fields*. Mathematics of Computation, **69**, 1267-1283, 2000.

[133] O. Schirokauer, D. Weber and T. Denny. *Discrete logarithms: the effectiveness of the index calculus method*. In Algebraic Number Theory Symposium (ANTS-II), Springer LNCS 1122, 337–361, 1996.

[134] C. P. Schnorr. *Efficient signature generation by smart cards*. J. Cryptology, **4**, 161–174, 1991.

[135] B. Schoenmakers and P. Tuyls, *Practical Two-Party Computation Based on the Conditional Gate.* In Advances in Cryptology (ASIACRYPT 2004), Springer LNCS 3329, 119–136, 2004.

[136] M. Scott. *Authenticated ID-based Key Exchange and remote log-in with insecure token and PIN number.* Cryptology ePrint Archive, Report 2002/164. Available from `http://eprint.iacr.org/2002/164`.

[137] M. Scott and P. Barreto. *Compressed Pairings.* In Advances in Cryptology (CRYPTO 2004), Springer LNCS 3152, 140–156, 2004.

[138] I. Semaev. *Summation polynomials and the discrete logarithm problem on elliptic curves.* Cryptology ePrint Archive, Report 2004/031. Available from `http://eprint.iacr.org/2004/031`.

[139] I. R. Shafarevich. *Basic Algebraic Geometry.* Springer Verlag, 1974.

[140] A. Shamir. *Identity-based cryptosystems and signature schemes.* In Advanced in Cryptology (CRYPTO 1984), Springer-Verlag LNCS 196, 47–53, 1984.

[141] D. Shanks. *The Infrastructure of Real Quadratic Field and its Applications.* In Proceedings of the 1972 Number Theory Conferecence: University of Colorado, Boulder, Colorado, August 14–18, 1972.

[142] V. Shoup. *Lower Bounds for Discrete Logarithms and Related Problems.* In Advances in Cryptology (EUROCRYPT 1997), Springer LNCS 1233, 256–266, 1997.

[143] V. Shoup. NTL – A Library for Number Theory. Available from: `http://www.shoup.net/ntl/`.

[144] J. Silverman. *The arithmetic of elliptic curves.* Springer GTM 106, 1986.

[145] N. P. Smart. *Cryptography: An Introduction.* McGraw-Hill, 2003.

[146] P. Smith and C. Skinner. *A public-key cryptosystem and a digital signature system based on the Lucas function analogue to discrete logarithms.* In Advances in Cryptology (ASIACRYPT 1995), Springer LNCS 917, 357–364, 1995.

[147] J. A. Solinas. *Low-Weight Binary Representations for Pairs of Integers.* University of Waterloo, Technical Report CORR 2001-41,

[148] M. Stam and A. K. Lenstra. *Efficient Subgroup Exponentiation in Quadratic and Sixth Degree Extensions.* In Cryptographic Hardware and Embedded Systems (CHES 2002), Springer LNCS 2523, 318–332, 2002.

[149] M. Stam and A. K. Lenstra. Speeding Up XTR. In Advances in Cryptology (ASIACRYPT 2001), Springer LNCS 2248, 125–143, 2001.

[150] E. G. Straus. *Problems and Solutions: (5125) Addition Chains of Vectors.* In American Mathematical Monthly, **71**, 806–808, 1964.

[151] E. Teske. *Speeding up Pollard's Rho Method for Computing discrete Logarithms.* In Algorithmic Number Theory Symposium (ANTS-III), Springer LNCS 1423, 541–554, 1998.

[152] The Certicom ECC Challenge: See `http://www.certicom.com/index.php`

[153] The Pairing-Based Crypto Lounge: See `http://paginas.terra.com.br/informatica/paulobarreto/pblounge.html`

[154] N. Thériault. *Index calculus attack for hyperelliptic curves of small genus.* In Advances in Cryptology (ASIACRYPT 2003), Springer LNCS 2894, 75–92, 2003.

[155] E. Thomé. *Computation of discrete logarithms in* $\mathbb{F}_{2^{607}}$. In Advances in Cryptology (AsiaCrypt 2001), Springer LNCS 2248, 107–124, 2001.

[156] M. van Dijk, R. Granger, D. Page, K. Rubin, A. Silverberg, M. Stam and D. Woodruff. *Practical cryptography in high dimensional tori*. In Advances in Cryptology (EUROCRYPT 2005), Springer LNCS 3494, 234–250, 2005.

[157] M. van Dijk and D. P. Woodruff. *Asymptotically optimal communication for torus-based cryptography*. In Advances in Cryptology (CRYPTO 2004), Springer LNCS 3152, 157–178, 2004.

[158] F. Vercauteren. *Computing zeta functions of cureves over finite fields.* PhD thesis, Katholieke Universiteit Leuven, 2003.

[159] F. Vercauteren and R. Lercier. *Discrete logarithms in* $GF(p^{18})$ *- 101 digits*: email to NMBRTHRY list, June 2005.

[160] E. Verheul. Personal Communication with Martijn Stam, 2001.

[161] V. E. Voskresenskiĭ. *Algebraic Groups and Their Birational Invariants.* Translations of Mathematical Monographs, **179**, American Mathematical Society, 1998.

[162] L. T. Yang and R. P. Brent. *The parallel improved Lanczos method for integer factorization over finite fields for public key cryptosystems.* In International Conference on Parallel Programming Workshops (ICPPW), IEEE Press, 106-114, 2001.