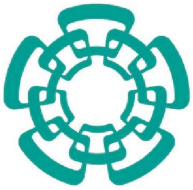

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS
AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO

DEPARTAMENTO DE COMPUTACIÓN

Logaritmo discreto en campos finitos de característica pequeña: atacando la criptografía basada en emparejamientos de Tipo 1

Tesis

Que presenta

Gora Adj

Para obtener el grado de

Doctor en Ciencias en Computatión

Directores de la Tesis:

Dr. Francisco José Rambó Rodríguez Henríquez

Dr. Alfred John Menezes

Ciudad de México Julio, 2016

A ma famille, spécialement à mes deux mamans.

Acknowledgements

I would like to start by thanking my advisors Francisco Rodríguez-Henríquez and Alfred Menezes for their clear guidance and constant support.

I also would like to thank all my friends and colleagues for having shared with me good times and hard times.

I am grateful to the Consejo Nacional de Ciencia y Tecnologia – CONACyT for the scholarship they provided me during all the period I was a PhD candidate at Cinvestav, and also for their financial support for the year I spent at University of Waterloo in Canada.

I thank the University of Waterloo, through Alfred Menezes, for their financial support during my visits the whole year of 2013 and in July 2014.

Thanks to ABACUS Supercompter – Cinvestav for opening their doors to us, which was very helpful to our latest computation project.

Part of Chapter 6 was realized while I was visiting Cinvestav in 2011-2012, from ISFA – Université Claude Bernard Lyon 1. I would like to thank Direction des Relations Internationales of Université Claude Bernard Lyon 1 for providing me a "bourse d'execellence" for my six-month stay at Cinvestav.

I also would like to thank Nareli Cruz-Cortés and the CONACyT (Project 132073) for their partial support.

Resumen

Los esquemas criptográficos basados en emparejamientos bilineales fueron propuestos en el área de criptografía asimétrica a principios de siglo, con la finalidad de dar solución a problemas que estuvieron abiertos por décadas. Uno de estos problemas, por nombrar sólo uno, consiste en cómo compartir un secreto entre tres entidades mediante el intercambio de información pública, en una sola ronda. Estos esquemas criptográficos utilizan un emparejamiento bilineal entre subgrupos cíclicos definidos en curvas elípticas o hiperelípticas, y un subgrupo multiplicativo en una cierta extensión del campo finito sobre el que se construyen las curvas.

Los emparejamientos bilineales simétricos conocidos como de Tipo 1 se definen sobre un campo base de característica 2 o 3, esto nos da la posibilidad de tomar ventaja de las características de la nueva generación de procesadores durante la fase de implementación debido a la naturaleza de la aritmética en estos campos, permitiendo así una implementación rápida y eficiente. Los emparejamientos de Tipo 1 brindan propiedades específicas que son aprovechadas en la construcción de novedosos protocolos basados en ellos.

Una condición de seguridad indispensable en un esquema criptográfico basado en emparejamientos, es que el problema del logaritmo discreto definido en los subgrupos de curvas y en el subgrupo del campo finito que componen el emparejamiento sea difícil de resolver. En los últimos años, ha habido avances significativos en el cálculo del logaritmo discreto en campos finitos de característica pequeña, lo que puso en una situación de incertidumbre e inestabilidad la seguridad de la criptografía basada en emparejamientos de Tipo 1.

En esta tesis, se demuestra por primera vez que los nuevos algoritmos para el cálculo del logaritmo discreto impactan drásticamente la seguridad de los esquemas basados en emparejamientos de Tipo 1. También, se muestra que los campos de característica pequeña $\mathbb{F}_{3^{6\cdot 509}}, \mathbb{F}_{2^{12\cdot 367}}, \mathbb{F}_{3^{6\cdot 1429}}$ y $\mathbb{F}_{2^{4\cdot 3041}}$ que se pensaba ofrecían 128 y 192 bits de seguridad en realidad brindad niveles de seguridad significativamente más bajos. El análisis que llevo a estas conclusiones fue posible gracias al diseño de herramientas y un marco de trabajo que permiten realizar evaluaciones prácticas de estos nuevos algoritmos. Además, se presenta la primera implementación de los recientes algoritmos para el cálculo del logaritmo discreto con el fin de atacar el campo finito $\mathbb{F}_{3^{6\cdot 137}}$, el cual es de interés criptográfico. Este cómputo ilustra la efectividad de los nuevos métodos para el cálculo del logaritmo discreto en campos finitos de característica pequeña que no sean de Kummer o *twisted*-Kummer. Debido a los refinamientos a los nuevos algoritmos, se realizó el primer cálculo de logaritmo discreto en un campo finito de característica 3, $\mathbb{F}_{3^{6\cdot 509}}$, que se pensaba ofrecía 128 bits de seguridad.

Además del trabajo realizado en el problema del logaritmo discreto, en esta tesis se presentan dos nuevos algoritmos que permiten calcular raíces cuadradas en extensiones de grado par de campos de característica grande. Uno de estos algoritmos supera a los algoritmos existentes en el estado del arte en varios escenarios criptográficos, mientras que el otro ofrece un mejor balance entre eficiencia y seguridad en comparación con los algoritmos existentes.

Abstract

Pairing-based protocols appeared in public key cryptography, at the beginning of the century, as solutions to problems that remained unsolved for decades. One of these problems, to name but one, is the possibility for three parties to share a secret key by exchanging information via a public network in only one round. These protocols essentially use bilinear pairing maps defined from cyclic subgroups of elliptic or hyperelliptic curves, to a subgroup of the multiplicative group of some extension of the finite field, over which are defined the curves.

In symmetric bilinear pairings, known as Type 1 pairings, the underlying finite field has characteristic 2 or 3. Besides the significant acceleration new generation processors can provide when performing operations in finite fields of characteristic 2 or 3, Type 1 pairings have the benefit of being equipped with specific properties that allow them to be employed in most pairing-based protocols.

A necessary condition for the security of a pairing-based cryptosystem is that the discrete logarithm problem (DLP) in the subjacent curve subgroups and the field subgroup must be hard. In recent years, there have been several dramatic improvements in algorithms for computing discrete logarithms in small characteristic finite fields, that consequently placed the security of the Type 1 pairing-based cryptography in a state of uncertainty.

In this thesis, we demonstrate for the first time that the new algorithms drastically impact the security of cryptosystems based on Type 1 pairings. We show that small characteristic finite fields of cryptographic interest, such as $\mathbb{F}_{3^{6\cdot 509}}$, $\mathbb{F}_{2^{12\cdot 367}}$, $\mathbb{F}_{3^{6\cdot 1429}}$, and $\mathbb{F}_{2^{4\cdot 3041}}$ that were assumed to enjoy 128 and 192 bits of security, in fact, find their security levels considerably lowered, and, at the same time, that of cryptographic protocols utilizing pairings derived from elliptic or hyperelliptic curves over these fields. The concrete analyses that led to these conclusions were made possible by designing a convenient framework and tools able to perform practical assessments of the new algorithms.

The first implementation of the new DLP algorithms for attacking a cryptographically interesting finite field, namely $\mathbb{F}_{3^{6\cdot 137}}$, is presented in this thesis. This computation illustrates the effectiveness of the new algorithms in small characteristic finite fields that are not Kummer or twisted-Kummer extensions.

As a consequence of more recent refinements on the recent methods, we completed the first discrete logarithm computation in a characteristic-three finite field that was previously believed to provide 128 bits of security, namely $\mathbb{F}_{3^{6\cdot 509}}$.

In addition to the work on the DLP over small characteristic finite fields, we present two new algorithms for computing square roots in even-degree extension fields of large characteristic. One of these algorithms outperforms previously existing algorithms in several cryptographic settings, while the other one offers a better trade-off between efficiency and security than previous methods.

Contents

1 Intoduction

The discrete logarithm problem (DLP) is one of the cryptologic problems that have been subjected to the most intensive research since the introduction of public key cryptography in 1976 by Diffie and Hellman [43]. A large number of cryptographic protocols devised since then rely for their security on the assumed intractability of the DLP in certain groups. Recall that the DLP in a group $G = \langle g \rangle$ denoted multiplicatively and of finite order N is the problem, given $h \in G$, of finding the integer $x \in [0, N-1]$ such that $h = g^x$. In this case, x is called the *discrete logarithm* of h to the base g and denoted $\log_g h$. The DLP in generic groups is known to be difficult [116]; the best algorithms known for finding discrete logarithms in such groups have running time $O(\sqrt{N})$ that is exponential in the bitlength of the group order.

Popular cryptographic choices of groups are subgroups of the multiplicative group of finite fields and subgroups of the additive group of points on either an elliptic curve or the jacobian of a hyperelliptic curve. Because of the underlying algebraic structure they provide, the DLP in these groups are in some instances easier than in generic groups. For example, over prime order fields $\mathbb{F}_p$, we have algorithms that have the same running time as algorithms for factoring integers of size that of p. These algorithms, which are of *subexponential* running time, are faster than any fully exponential time algorithm. In some families of elliptic and hyperelliptic curves, we also know algorithms running in subexponential time. However, in the specific case of elliptic curves, for the majority of suitable cryptographic groups, the best algorithm for solving the DLP is of fully exponential complexity.

Now regarding non-prime finite fields $\mathbb{F}_{p^n}$, with the *characteristic* p of medium to small size, the Waterloo improvement in 1983 by Blake, Fuji-Hara, Mullin and Vanstone [28] on Adleman's subexponential time algorithm [8] was the first result to demonstrate that the DLP in these fields is not in practice as difficult as in prime fields, although the asymptotic complexity remained unchanged. Shortly thereafter, in 1984 Coppersmith [40] strengthened this presumption by presenting an algorithm that to date is faster than any algorithm attacking the DLP in prime fields. Coppersmith used his algorithm to compute discrete logarithms in the field $\mathbb{F}_{2^{127}}$.

1.1 Motivation

More recently, a whole series of dramatic breakthroughs have been published in 2013. The new algorithms, discovered by Joux [77], Göloğlu et al. [57, 58] and Barbulescu et al. [16], dramatically improved the complexity of solving the DLP in finite fields of small characteristic. Joux and Göloğlu et al. used their algorithms to effectively compute discrete logarithms in $\mathbb{F}_{2^{8\cdot 3\cdot 255}} = \mathbb{F}_{2^{6120}}$ in 750 CPU hours [58], and in $\mathbb{F}_{2^{8\cdot 3\cdot 257}} = \mathbb{F}_{2^{6168}}$ in 550 CPU hours [78]. In spite of the striking aspects of these calculations, in terms of both the small expended

effort and the huge size of the fields, one should notice the special nature of the fields $\mathbb{F}_{2^{6120}}$ and $\mathbb{F}_{2^{6168}}$. Indeed, $\mathbb{F}_{2^{6120}}$ is a degree-255 extension of $\mathbb{F}_{2^{8\cdot 3}}$ with $255 = 2^8 - 1$ (a Kummer extension), and $\mathbb{F}_{2^{6168}}$ is a degree-257 extension of $\mathbb{F}_{2^{8\cdot 3}}$ with $257 = 2^8 + 1$ (a twisted Kummer extension).

The new improvements are potentially relevant to the security of Type 1 pairing-based cryptosystems that use bilinear pairings derived from supersingular elliptic curves or genus-2 hyperelliptic curves defined over finite fields of characteristic 2 or 3. In this setting, one is concerned with the DLP in finite fields $\mathbb{F}_{2^{4\cdot n}}$, $\mathbb{F}_{2^{12\cdot n}}$ and $\mathbb{F}_{3^{6\cdot n}}$, where n is prime; note that these fields are not Kummer or twisted Kummer extensions and so the special properties of such extensions cannot be exploited.

The introduction of bilinear pairings to cryptography dates back to 1991. They initially were employed to attack cryptosystems that used supersingular elliptic curves [100, 49] by exploiting their ability of mapping a pair of points on an elliptic curve to the multiplicative group of a finite field and thus reducing the DLP on elliptic curves to the much more easy case of finite fields. In 2000, bilinear pairings started generating renewed interest when they appeared as the core ingredient in new cryptographic protocols proposed as solutions to problems that had remained open for decades. This gave rise to the field of pairing-based cryptography. One of the most interesting protocols among these is the Boneh-Franklin identity-based encryption scheme [30], where Alice does not need to generate a pair of private/public keys before Bob can encrypt a message for her; instead Bob simply uses Alice's identity, for example Alice's email address. By 2004, there were already more than 200 papers published on pairing-based cryptography.

Type 1 parings, also called *symmetric* pairings, are one of the three main families of bilinear pairings. In fact, cryptographic protocols employing pairings are most commonly described in the Type 1 setting because of its relative simpler mathematical structure. From the introduction of paring-based cryptography to 2012, the cryptographic parameters, when designing paring-based cryptosystems using Type 1 pairings arising from supersingular curves over small characteristic fields, were chosen under the assumption that Coppersmith's algorithm is the fastest method for finding discrete logarithms in finite fields of small characteristic.

The purpose of this thesis is to demonstrate that the new DLP algorithms can be used to render the DLP in finite field subgroups arising from Type 1 pairings much easier than previously believed and at the same time show that Type 1 pairings are no longer suitable for pairing-based cryptography. Furthermore, we study the problem of computing square roots in large prime field extensions, which is of great interest in several pairing-based protocols devised over elliptic curves, with the specific goal of providing improvements on the calculation of square roots in even-degree extension fields.

1.2 Contributions

Concrete analysis in cryptographic finite fields [3, 4]. We combine the new algorithms to show that the finite fields $\mathbb{F}_{3^{6\cdot 509}}$, $\mathbb{F}_{2^{12\cdot 367}}$ and $\mathbb{F}_{2^{4\cdot 3041}}$ are weak for discrete logarithm cryptography in the sense that discrete logarithms in these fields can be computed significantly

faster than with the previous fastest algorithms. Our concrete analyses show that the supersingular curves with embedding degree 6, 12 and 4 defined, respectively, over $\mathbb{F}_{3^{509}}$, $\mathbb{F}_{2^{367}}$ and $\mathbb{F}_{2^{3041}}$ that had been considered for implementing Type 1 pairing-based cryptosystems in fact provide significantly lower levels of security. Moreover, our work provides a convenient framework and tools for performing a concrete analysis of the new discrete logarithm algorithms and their variants.

Analysis and discrete logarithm computations in cryptographic finite fields [4, 5]. We use the polynomial representation of Granger and Zumbrägel [64] to examine the effectiveness of the new algorithms for computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 1429}}$, and to show that a Magma implementation of Joux's algorithm can be used to compute discrete logarithms in the 1303-bit finite field $\mathbb{F}_{3^{6\cdot 137}}$ and the 1551-bit finite field $\mathbb{F}_{3^{6\cdot 163}}$ with very modest computational resources. The intractability of the discrete logarithm problem in these fields is necessary for the security of Type 1 pairings derived from supersingular elliptic curves with embedding degree 6. The elliptic curve over $\mathbb{F}_{3^{1429}}$ was believed to enjoy a security level of 192 bits against attacks by Coppersmith's algorithm. Our analysis shows that this curve offers a security level of at most 96 bits. Our $\mathbb{F}_{3^{6\cdot 137}}$ implementation was the first to illustrate the effectiveness of Joux's algorithm for computing discrete logarithms in small characteristic finite fields that are not Kummer or twisted-Kummer extensions.

Discrete logarithm computations at a (previously assumed) 128-bit security level [5]. We show that the techniques from [60], [61] and [82] lower our previous estimates for computing discrete logarithms in the 4841-bit characteristic-three field $\mathbb{F}_{3^{6\cdot 509}}$ from 81.7 bits to 58.9 bits. An immediate consequence of this drop in security level is that we completed the computation of discrete logarithms in the order-r subgroup of $\mathbb{F}^*_{3^{6\cdot 509}}$, where $r = (3^{509} - 3^{255} + 1)/7$ is an 804-bit prime, within 220 CPU years. Recall that the previously believed intractability of the DLP in $\mathbb{F}_{3^{6\cdot 509}}$ was a strong security argument for the hardness of the elliptic curve discrete logarithm problem in $E(\mathbb{F}_{3^{509}})$, where E is the supersingular elliptic curve $y^2 = x^3 - x + 1$ with $|E(\mathbb{F}_{3^{509}})| = 7r$. In addition, we use techniques from [60] to reduce the estimates for computing discrete logarithms in the 13590-bit characteristic-three field $\mathbb{F}_{3^{6\cdot 1429}}$ from 95.8 bits to 78.8 bits.

Two new algorithms for computing square roots in even-degree extension fields [6]. The computation of square roots in extension fields of the form $\mathbb{F}_{q^2}$, with $q = p^n$, p a large odd prime and $n \geq 1$, is an operation frequently performed in a series of asymmetric pairing-based cryptosystems using elliptic curves such as the Barreto-Naehrig curves. We present two new algorithms for the cases where $q \equiv 1 \pmod 4$ and $q \equiv 3 \pmod 4$. From the complexity analysis of these algorithms and the corresponding experimental results we provide, one can see that our new algorithms are competitive against the *complex method* of Scott [113]. Moreover, we present a procedure that tests squareness of elements in an extension field $\mathbb{F}_{p^n}$, p a large prime number, at the cost of several applications of the inexpensive p-powering operation plus the computation of the Legendre symbol in the prime field $\mathbb{F}_p$. This procedure is faster than the recursive algorithm proposed by Bach and Huber in [13].

1.3 Outline

The remainder of this document is organized as follows. In Chapter 2, we provide some helpful definitions and fundamental results on the underlying theory and concepts of this thesis. Chapter 3 presents our concrete analyses on the discrete logarithm computation in the finite fields $\mathbb{F}_{3^{6\cdot 509}}$, $\mathbb{F}_{2^{12\cdot 367}}$ and $\mathbb{F}_{2^{4\cdot 3041}}$. In Chapter 4, we describe the experimental computations of discrete logarithms in the fields $\mathbb{F}_{3^{6\cdot 137}}$ and $\mathbb{F}_{3^{6\cdot 163}}$, and the analysis of the DLP in $\mathbb{F}_{3^{6\cdot 1429}}$. Our ongoing work on computing discrete logarithms in the 4841-bit field $\mathbb{F}_{3^{6\cdot 509}}$ is outlined in Chapter 5, together with our latest estimates for discrete logarithm computations in the 13590-bit field $\mathbb{F}_{3^{6\cdot 1429}}$. Chapter 6 presents the study on square root computations in extension fields. Finally, Chapter 7 provides concluding remarks on the work that has been conducted in this thesis before looking forward to some avenues for future research.

2 Basic Concepts

2.1 Mathematical background

In this section, we recall some definitions and results on algebraic objects that are frequently used in this document. For more explicit details in the present content, we refer the reader to classical text-books on commutative algebra [11, 96, 105], algebraic geometry [68, 117] and public-key cryptography [39, 88, 86, 122].

2.1.1 Groups, rings and fields

Definition 2.1. A *group* $(G, *)$ is a non-empty set G together with a mapping $* : G \times G \to G$, called *composition law*, satisfying the three following properties:

- *Associativity*: for all $a, b, c \in G$, we have $(a * b) * c = a * (b * c)$.
- *Identity element*: there exists $e \in G$ such that for all $a \in G$, $a * e = e * a = a$.
- *Inverse element*: for all $a \in G$, there exists $a' \in G$ such that $a * a' = a' * a = e$.

Let $(G, *)$ be a group. If for all $a, b \in G$, we have $a * b = b * a$, then $(G, *)$ is said to be *commutative* or *abelian*. $(G, *)$ is called *cyclic* if there exists an element $g \in G$ such that all element $b \in G$ can be written as $b = a * \ldots * a$ (n times, for some integer n); the element g is then called a *generator* of G and we write $G = \langle g \rangle$. Let H be a subset of G. If $(H, *)$ itself is a group then it is said to be a *subgroup* of $(G, *)$. Often we use G as a short name for the group $(G, *)$.

A group is said to be *finite* if it contains finitely many elements. The number of elements in a finite group is called its *order*. We shall write $|G|$ for the order of a finite group G.

Definition 2.2. A *group homomorphism* between two groups $(G, *)$ and $(G', \bullet)$ is a map $\psi : G \to G'$ such that for all $a, b \in G$: $\psi(a * b) = \psi(a) \bullet \psi(b)$. If the homomorphism is a one-to-one correspondence between G and G' then it is called *isomorphism* and the groups are said to be *isomorphic*.

Definition 2.3. Let H be a subgroup of an abelian group $(G, \cdot)$. Let $a \in G$, the *coset* of H in G with respect to a is the set $aH = \{a \cdot b \mid b \in H\}$. The set of all cosets of H in G is defined to be $G/H = \{aH \mid a \in G\}$. For $a, b \in G$, we define the composition law $*$ on G/H as $(aH) * (bH) = (ab)H$.

It is easy to verify that G/H together with the law defined in the above definition is an abelian group, we call it *quotient group*.

Definition 2.4. A *ring* $(R, +, \times)$ is a non-singleton non-empty set R together with two composition laws, called *addition* and *multiplication*, $+, \times : R \times R \to R$ such that:

- $(R, +)$ is an abelian group, with identity element denoted 0.
- $R^* = R \backslash \{0\}$ with respect to $\times$ is associative and has an identity element denoted 1.
- The *distributivity laws* hold, that is, for all $a, b, c \in R$, we have $a \times (b+c) = (a \times b) + (a \times c)$ and $(b + c) \times a = (b \times a) + (c \times a)$.

If $(R, +, \times)$ is a ring, we often use R as a short name for it. A ring R is said to be *commutative* if the law $\times$ is commutative, that is, for all $a, b \in R$, $a \times b = b \times a$.

Definition 2.5. Let $(R, +, \times)$ and $(R', \oplus, \otimes)$ be two rings. A *ring homomorphism* is a map $\psi : R \to R'$ such that for all $a, b \in R$:

- $\psi(a + b) = \psi(a) \oplus \psi(b)$.
- $\psi(a \times b) = \psi(a) \otimes \psi(b)$.
- $\psi(1) = 1$.

Definition 2.6. A subset I of a ring $(R, +, \times)$ is called *ideal* of R if I is a subgroup of $(R, +)$ and for all $f \in I$ and $a \in R$, $(f \times a) \in I$ and $(a \times f) \in I$. In this case, I is said to be *prime* if for all $f, g \in R$ such that $f \times g \in I$, then $f \in I$ or $g \in I$.

It can be verified that for any element f in a commutative ring R, the set $\{a \times f, a \in R\}$ is an ideal of R, denoted (f). Also, given a ring R and an ideal I of R, the set of equivalence classes, with respect to the relation $a \sim b$ if and only if $(a - b) \in I$, forms a ring called a *quotient ring* and denoted $(R/I, +, \times)$. The elements of R/I will be viewed as the elements of R, with the understanding that if $(a - b) \in I$, where $a, b \in R$, then we write $a = b$ in R/I.

Definition 2.7. A commutative ring is called *integral domain* if the product of any two nonzero elements is nonzero.

One can easily show that the quotient ring of a commutative ring by a prime ideal of it is an integral domain.

Definition 2.8. A *field* is a ring $(R, +, \times)$ such that $(R^*, \times)$ is an *abelian group*. A *finite field*, also known as *Galois field*, is a field that contains finitely many elements. The number of elements in a finite is called its *order*.

A subset $\mathbb{K}$ of a field $(\mathbb{F}, +, \times)$ is called *subfield* of $\mathbb{F}$ if $(\mathbb{K}, +, \times)$ itself is a field. In this case, $\mathbb{F}$ is called an *extension field* of $\mathbb{K}$.

Definition 2.9. Let $\mathbb{F}$ and $\mathbb{K}$ be two fields. A *homomorphism of fields* $\psi : \mathbb{F} \to \mathbb{K}$ is a ring homomorphism between $\mathbb{F}$ and $\mathbb{K}$. We say that the homomorphism of fields ψ is an *isomorphism* if it is a one-to-one correspondence between $\mathbb{F}$ and $\mathbb{K}$. In this case the two fields are said to be *isomorphic*.

One can easily see that a homomorphism of fields is always one-to-one. For a positive integer q, it is well known that there exists a finite field of order q if and only if q is a power of a prime number called the *characteristic* of the field. Furthermore, any two finite fields with same order are isomorphic. Therefore, for q a power of a prime, there is a unique up to isomorphism finite field of order q, denoted $\mathbb{F}_q$. We also have that an extension of the field $\mathbb{F}_q$ is necessarily of the form $\mathbb{F}_{q^n}$, with n some nonzero positive integer. Another well known result on finite fields is that the multiplicative group $\mathbb{F}_q^*$, for q a power of a prime, is always a cyclic group, where a generator is called *primitive element*.

Let $\mathbb{F}$ be a field. It can be shown that the set of all polynomials in variables $X_1, X_2, \ldots, X_k$ with coefficients in $\mathbb{F}$ is a ring, denoted $\mathbb{F}[X_1, X_2, \ldots, X_k]$ and called a *polynomial ring* over $\mathbb{F}$. Now let $f(X) \in \mathbb{F}_q[X]$ be an irreducible polynomial over a finite field $\mathbb{F}_q$ (that is, whenever $f(X)$ is written $f(X) = g(X)h(X)$ then $g(X) \in \mathbb{F}_q$ or $h(X) \in \mathbb{F}_q$) of degree a positive integer n. Then it turns out that the quotient ring $\mathbb{F}_q[X]/(f(X))$ is a finite field of order q^n. Thus, we have $\mathbb{F}_{q^n} \cong \mathbb{F}_q[X]/\left(f(X)\right)$ and the elements of $\mathbb{F}_{q^n}$ can be represented by polynomials over $\mathbb{F}_q$ of degree at most $n-1$, where the addition of elements is the standard addition of polynomials and the multiplication is the polynomial multiplication modulo $f(X)$.

Let R be an integral domain. Consider the equivalence relation: for $a, b, c, d \in R,\ b \neq 0, d \neq 0,\ (a, b) \sim (c, d)$ if and only if $ad = bc$. The set of equivalence classes on R with respect to $\sim$ forms a field called *fraction field* and denoted by $\mathrm{Frac}(R)$. An element of $\mathrm{Frac}(R)$ represented by (a, b), $b \neq 0$, is denoted by a/b.

Definition-proposition 2.10. Let $\mathbb{F}$ be a field. If $\mathbb{F}$ has the property that any polynomial in $\mathbb{F}[X]$ can be written as a product of polynomials in $\mathbb{F}[X]$ of degree at most 1, then we say that $\mathbb{F}$ is *algebraically closed*. Among all the algebraically closed extensions of $\mathbb{F}$ there is a unique up to isomorphism smallest extension, called the *algebraic closure* of $\mathbb{F}$ and denoted $\overline{\mathbb{F}}$.

2.1.2 Elliptic and hyperelliptic curves over finite fields

2.1.2.1 Elliptic curves

Definition 2.11. Let $\mathbb{F}$ be a field. An *elliptic curve* E over $\mathbb{F}$ is a curve satisfying the following properties:

- E is given by the Weierstrass equation defined as follows

$$y^2 + a_1xy + a_3y = x^3 + a_2y^2 + a_4x + a_6, a_i \in \mathbb{F}. \tag{2.1}$$

- The two equations

$$a_1 y = x^3 + 2a_2 x + a_4, \qquad 2y + a_1 y + a_3 = 0 \tag{2.2}$$

  together with (2.1) cannot be simultaneously satisfied by any pair $(u, v) \in \overline{\mathbb{F}}^2$.

For any extension field $\mathbb{K}$ of $\mathbb{F}$, we denote by $E(\mathbb{K})$ the set of $(u, v) \in \mathbb{K}^2$ satisfying (2.1), along with a *point at infinity* denoted by O.

An important fact about the set of points of an elliptic curve is that they form an abelian group with the point at infinity as identity element. The group law, denoted additively, is typically constructed geometrically over the real plane, that is, when the elliptic curve is defined over $\mathbb{R}$. However, the formulas obtained over the real plane from the Weierstrass equation (2.1) can be systematically exported to any field.

Theorem 2.12. Let $\mathbb{F}$ be a field and E an elliptic curve over $\mathbb{F}$ given by equation (2.1). The additive law "+" described as follows: for all $Q = (u_1, v_1)$, $R = (u_2, v_2) \in E(\mathbb{F})$,

- $-Q = (u_1, -v_1 - a_1 u_1 - a_3)$.
- $Q + R = O$, if $R = -Q$.
- $Q + R = (u_3, v_3) = (\lambda^2 + a_1\lambda - a_2 - u_1 - u_2, \lambda(u_1 - u_3) - v_1 - a_1 u_3 - a_3)$, where

$$\lambda = \begin{cases} (v_1 - v_2) \,/\, (u_1 - u_2) & \text{if } Q \neq \pm R, \\ (3u_1^2 + 2a_2 u_1 + a_4 - a_1 v_1) \,/\, (2v_1 + a_1 u_1 + a_3) & \text{if } Q = R \end{cases} \tag{2.3}$$

defines an abelian group over $E(\mathbb{F})$.

Except for the associativity, one can easily check that this law has all the required properties to be an abelian group. The associativity can be verified by a lengthy computation using explicit formulas, or by using more advanced algebraic or analytic methods. A complete proof can be found in [122].

Definition 2.13. Let E be an elliptic curve over $\mathbb{F}_q$, with $\mathbb{F}_q$ of characteristic p. For all $k \in \mathbb{N}$ and $P \in E\left(\overline{\mathbb{F}}\right)$, we define $[k]P = \underbrace{P + \cdots + P}_{\text{k times}}$ if $k \neq 0$ and $[0]P = O$. The elliptic curve E is said to be *supersingular* if there exists $r \in \mathbb{N}$ such that the following identity holds:

$$\{P \in E\left(\overline{\mathbb{F}}\right) \mid [p^r]P = O\} = \{O\}.$$

Otherwise, E is said to be *ordinary*.

2.1.2.2 Hyperelliptic curves

Definition 2.14. Let $\mathbb{F}$ be a field. A *hyperelliptic curve* C of *genus* $g \in \mathbb{N}$ over $\mathbb{F}$ is a curve satisfying the following properties:

- C is given by the following equation

$$y^2 + h(x)y = f(x), \tag{2.4}$$

  where $h(x) \in \mathbb{F}[x]$ is of degree at most g and $f(x) \in \mathbb{F}[x]$ is monic of degree $2g+1$.

- The two equations

$$2y + h(x) = 0, \qquad h'(x)y - f'(x) = 0, \tag{2.5}$$

  together with (2.4) cannot be simultaneously satisfied by any pair $(u, v) \in \overline{\mathbb{F}}^2$.

For any extension field $\mathbb{K}$ of $\mathbb{F}$, we denote by $C(\mathbb{K})$ the set of points $(u, v) \in \mathbb{K}^2$ satisfying (2.4) along with a *point at infinity* denoted by O. We call *finite points* the points of C different of the point at infinity. The curve $C\left(\overline{\mathbb{F}}\right)$ is simply denoted by C.

Definition 2.15. Let $P = (u, v)$ be a finite point on a hyperelliptic curve C. The *opposite* of P is the point $\overline{P} = (u, -v - h(u))$, which is also on C. The opposite of O is O itself. If P satisfies $P = \overline{P}$, then it is said to be *special*; otherwise it is said to be *ordinary*.

Definition 2.16. Let C be a hyperelliptic curve given by equation (2.4) over a field $\mathbb{F}$. The *coordinate ring* of C, denoted by $\mathbb{F}[C]$, is the quotient ring

$$\mathbb{F}[C] = \mathbb{F}[x, y]/\big(y^2 + h(x)y - f(x)\big).$$

An element of $\overline{\mathbb{F}}[C]$ is called *polynomial function* on C.

By definition, $y^2 = f(x) - h(x)y$ in $\mathbb{F}[C]$. Then, we can repeatedly replace any occurrence of y^2 in all polynomial function $G(x, y) \in \overline{\mathbb{F}}[C]$ by $f(x) - h(x)y$ and end up with a representation as $G(x, y) = a(x) - b(x)y$, where $a(x), b(x) \in \overline{\mathbb{F}}[x]$. It can be verified that the ideal $\big(y^2 + h(x)y - f(x)\big)$ is prime in $\mathbb{F}[x, y]$. Thus, $\mathbb{F}[C]$ is an integral domain.

Definition 2.17. The *function field* $\mathbb{F}(C)$ of a hyperelliptic curve C over a field $\mathbb{F}$ is the field of fractions of $\mathbb{F}[C]$. The elements of $\mathbb{F}(C)$ are called *rational functions* on C.

Definition 2.18. Let C be a hyperelliptic curve over a field $\mathbb{F}$. Let $R \in \overline{\mathbb{F}}(C)$ and $P = (u, v) \in C$ be a finite point. The rational function R is said to be *defined at* P if there exist polynomial functions $G, H \in \overline{\mathbb{F}}[C]$ such that $R = G/H$ and $H(P) = H(u, v) \neq 0 \in \overline{\mathbb{F}}$. In this case, the *value* of R at P is

$$R(P) = G(P)/H(P) = G(u, v)/H(u, v).$$

If no such G, H exist, then R is said to be *not defined at* P.

It is easy to see in the above definition that the value $R(P)$ of R at P does not depend on the choice of G and H.

Definition 2.19. Let $G(x, y) = a(x) - b(x)y \in \overline{\mathbb{F}}[C]$ be a polynomial function on a hyperelliptic curve C of genus g over a field $\mathbb{F}$. The *degree* of G is defined to be

$$\deg(G) = \max\{2\deg_x(a), 2g + 1 + 2\deg_x(b)\},$$

where $\deg_x(\cdot)$ denotes the degree with respect to the variable x.

Definition 2.20. Let $R = G/H \in \overline{\mathbb{F}}(C)$ be a rational function on a hyperelliptic curve C over a field $\mathbb{F}$.

- if $\deg(G) < \deg(H)$ then R is *defined at* O and the *value* of R at O is $R(O) = 0$,
- if $\deg(G) > \deg(H)$ then R is *not defined at* O,
- if $\deg(G) = \deg(H)$ then R is *defined at* O and the *value* of R at O is defined to be the ratio of the leading coefficient of G over the leading coefficient of H.

Here also, one can easily see that the value $R(P)$ of R at O does not depend on the choice of G and H.

Definition 2.21. Let $R \in \overline{\mathbb{F}}(C)$ be a rational function on a hyperelliptic curve C over a field $\mathbb{F}$, and let $P \in C$. If $R(P) = 0$ then R is said to have a *zero* at P. If R is not defined at P then R is said to have a *pole* at P.

Definition 2.22. Let $G(x, y) = a(x) - b(x)y \in \overline{\mathbb{F}}[C]$ be a nonzero polynomial function on a hyperelliptic curve C over a field $\mathbb{F}$. The *order of* G at P, denoted by $\mathrm{ord}_P(G)$, is defined as follows:

- If $P = (u, v)$ is a finite point, then
  - $\mathrm{ord}_P(G) = r + s$, if P is an ordinary point;
  - $\mathrm{ord}_P(G) = 2r + s$, if P is a special point,

  where r is the highest power of $(x-u)$ that divides both $a(x)$ and $b(x)$, written $G(x, y) = (x - u)^r(a_0(x) - b_0(x)y)$, and s is
  - 0, if $(a_0(u) - b_0(u)v) \neq 0$;
  - the highest power of $(x - u)$ that divides $N((a_0(x) - b_0(x)y)$, otherwise.
- If $P = O$, then $\mathrm{ord}_P(G) = -\deg(G)$.

An important result on the finiteness of the number of zeros and poles of a polynomial function is given in the following theorem.

Theorem 2.23. Let $G(x, y) \in \overline{\mathbb{F}}[C]$ be a nonzero polynomial function on a hyperelliptic curve C over a field $\mathbb{F}$. Then G has a finite number of zeros and poles. Moreover, $\sum_{P \in C} \operatorname{ord}_P = 0$.

Definition 2.24. Let $R = G/H \in \overline{\mathbb{F}}(C)$ be a rational function on a hyperelliptic curve C over a field $\mathbb{F}$. The *order* of R at P is defined to be $\operatorname{ord}_P(R) = \operatorname{ord}_P(G) - \operatorname{ord}_P(H)$.

It can be verified in the above definition that $\operatorname{ord}_P(R)$ does not depend on the choice of G and H. Theorem 2.23 is also true for nonzero rational functions.

Definition 2.25. A *divisor* D on a hyperelliptic curve C is a formal sum of points on C:

$$D = \sum_{P \in C} m_P P, \quad m_P \in \mathbb{Z} \ \ \forall P \in C,$$

where only a finite number of the m_P are nonzero. The *degree* of D, denoted $\deg D$, is the integer $\sum_{P \in C} m_P$. The *order* of D at a point $P \in C$, denoted $\operatorname{ord}_P(D)$, is the integer m_P.

The set of all divisors on a hyperelliptic curve C is denoted by $\mathbb{D}$. It can be easily shown that $\mathbb{D}$ is an abelian group under the additive law:

$$\sum_{P \in C} m_P P + \sum_{P \in C} n_P P = \sum_{P \in C} (m_P + n_P) P.$$

The set of all divisors in $\mathbb{D}$ of degree 0, denoted by $\mathbb{D}^0$, is a subgroup of $\mathbb{D}$.

Definition 2.26. Let $R \in \overline{\mathbb{F}}(C)$ be a rational function on a hyperelliptic curve C over a field $\mathbb{F}$. The *divisor of* R is defined to be

$$\operatorname{div}(R) = \sum_{P \in C} \operatorname{ord}_P P.$$

Theorem 2.23 shows that the divisor of a rational function R on a hyperelliptic curve C is indeed a divisor on C and we have $\operatorname{div}(R) \in \mathbb{D}^0$. It can be shown that the set of divisors of the rational functions over C form a subgroup of $\mathbb{D}^0$.

Definition 2.27. Let C be a hyperelliptic curve over a field $\mathbb{F}$. A divisor $D \in \mathbb{D}^0$ is called *principal divisor* if $D = \operatorname{div}(R)$. Denoting by $\mathbb{P}$ the subgroup of $\mathbb{D}^0$ of all the principal divisor on C, we define the *jacobian* of C to be the quotient group $\mathbb{J} = \mathbb{D}^0/\mathbb{P}$.

For an extension field $\mathbb{K}$ of $\mathbb{F}$, we also define $\mathbb{J}(\mathbb{K}) = \mathbb{D}^0(\mathbb{K})/\mathbb{P}(\mathbb{K})$, with $\mathbb{D}^0(\mathbb{K})$ and $\mathbb{P}(\mathbb{K})$ the respective subgroups of $\mathbb{D}^0$ and $\mathbb{P}$ where the points appearing in a divisor are restricted to those lying on $C(\mathbb{K})$.

Definition 2.28. Let C be a hyperelliptic curve of genus g over a finite field $\mathbb{F}_q$. The jacobian $\mathbb{J}$ of C is said to be *supersingular* if there exists a positive integer k such that

$$|\mathbb{J}(\mathbb{F}_{q^{2k}})| = \left(q^k \pm 1\right)^{2g}.$$

A hyperelliptic curve is said to be *supersingular* if its jacobian is supersingular; otherwise it is said to be *ordinary*.

2.2 Type 1 pairings

2.2.1 Bilinear pairing types

Definition 2.29. Let $(\mathbb{G}_1, +)$, $(\mathbb{G}_2, +)$ and $(\mathbb{G}_T, \cdot)$ be three cyclic groups of the same prime order. A *pairing* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate bilinear map, that is:

- $e(Q, R) = 1$ for all $R \in \mathbb{G}_2$ implies $\mathbb{G}_1 \ni Q = 0$.
- $e(Q_1 + Q_2, R) = e(Q_1, R) \cdot e(Q_2, R)$ for all $Q_1, Q_2 \in \mathbb{G}_1$ and $R \in \mathbb{G}_2$.
- $e(Q, R_1 + R_2) = e(Q, R_1) \cdot e(Q, R_2)$ for all $Q \in \mathbb{G}_1$ and $R_1, R_2 \in \mathbb{G}_2$.

In cryptographic applications, where they need to be efficiently computable, bilinear pairings are typically constructed over elliptic or hyperelliptic curves.

Cryptographic bilinear pairings are classified into different types. This classification, due to Galbraith, Paterson and Smart [53], is based on the structure of the underlying groups:

- **Type 1**: when $\mathbb{G}_1 = \mathbb{G}_2$, the pairing is said to be symmetric.
- **Type 2**: when the pairing is asymmetric, that is, $\mathbb{G}_1 \neq \mathbb{G}_2$, and we know an efficiently computable isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$.
- **Type 3**: when the pairing is asymmetric and no efficiently computable isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ is known.

While Type 3 pairings are not equipped with known efficiently computable isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$, they provide efficient ways to map a message to a point in $\mathbb{G}_2$ (called hash into $\mathbb{G}_2$) which do not exist in a Type 2 pairings. Hashing to $\mathbb{G}_2$ is also efficient in Type 1 pairings.

Type 1 pairings are derived from supersingular curves (elliptic or hyperelliptic) which can be divided into two groups: those defined over small characteristic (2 or 3) finite fields and those defined over large characteristic fields. Types 2 and 3 are in turn derived from ordinary curves defined over finite fields over large characteristic. In all of these types $\mathbb{G}_T$ is a multiplicative subgroup of an extension of the underlying finite field of the curves.

When implementing cryptographic protocols using parings, Types 2 and 3 are considered to be better choices than Type 1 in terms of efficiency. The main reason for this preference is a larger bitlength of elements of $\mathbb{G}_1$ in Type 1, making therefore operations in $\mathbb{G}_1$ more costly.

However, there are pairing-based protocols in the literature, such as the Boneh-Shacham group signature scheme [32], where one needs to apply an efficiently computable isomorphism

to a point obtained after a hashing into $\mathbb{G}_2$. To implement these protocols neither Type 2 nor Type 3 can be used whereas Type 1 provides simultaneously all the required properties. Also recall that instructions in new generation processors such as the carry-less multiplication in Intel machines can significantly accelerate operations in small characteristic finite fields, hence making small characteristic Type 1 pairings very attractive.

2.2.2 Type 1 pairing-based protocols

In this section, we give a brief description of three fundamental examples of Type 1 pairing-based protocols. Note that there is a much longer list of interesting and practical cryptographic schemes using Type 1 bilinear pairings, but these three are the most illustrative protocols demonstrating the importance of bilinear parings. For all three protocols, one can see that a necessary condition for security is the hardness of the DLP in the group $\mathbb{G}_T$. For pairings derived from elliptic and hyperelliptic curves over a finite field $\mathbb{F}_q$, $\mathbb{G}_T$ is a prime-order subgroup of the multiplicative group of an extension field $\mathbb{F}_{q^k}$.

2.2.2.1 Three-party one-round key agreement

The Diffie-Hellman key agreement scheme allows two parties to share in one round a secret that can be later used to securely exchange messages over a public network. In 2000, Joux [75] showed that using pairings, one can extend the one-round key agreement scheme to three parties.

Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ be a Type 1 pairing, with $r = |\mathbb{G}_1| = |\mathbb{G}_T|$ and $\mathbb{G}_1 = \langle P \rangle$. In Joux's scheme, Alice randomly selects a secret integer $a \in [1, r-1]$ and send the point $[a]P$ to Bob and Chris via a public network. Simultaneously, Bob sends $[b]P$ to Alice and Chris, and Chris sends $[c]P$ to Alice and Bob. After all the points are received, Alice computes $K = e([b]P, [c]P)^a = e(P, P)^{abc}$, and Bob and Chris do the same to get the shared key K. An eavesdropper who wishes to compute K needs to solve the bilinear Diffie-Hellman problem (BDHP): given $P,\ [a]P,\ [b]P,\ [c]P$, compute $e(P, P)^{abc}$.

2.2.2.2 Identity-based encryption

Traditional public-key encryption schemes rely on a certifying authority for providing certificates for public keys. In this way, Bob is certain that he uses an authentic copy of Alice's public key when sending encrypted messages to Alice, and not the public key of an attacker who would thereafter be able to decrypt Bob's messages that were intended only for Alice. However, in this scenario, Bob does not have the assurance that Alice's public key is still valid, because her certificate may have been revoked by the certifying authority on account of Alice having left her place of employment or her private key somehow having been compromised. In 1984, Shamir introduced the notion of a public key encryption scheme in which the public key can be an arbitrary string, with the motivation of simplifying certificate management in e-mail systems. Shamir's proposition is that when Bob needs to send a message to Alice, he simply uses Alice's email address (or any other identifying information of Alice) as Alice's public key and the public key of a trusted third party (TTP). When Alice receives Bob's message, she contacts the TTP from which she obtains her private key and decrypts Bob's message. Notice that, unlike the case with traditional certificate-based encryption schemes,

it is not necessary for Alice to generate a key pair before Bob can encrypt a message for her. The key revocation problem inherent with traditional certificates can be circumvented by adding a date to Alice's email address when encrypting; the TTP would only give to Alice the corresponding private key if it has not been revoked by that date.

Since Shamir only gave a framework for identity-based encryption, the problem of designing an identity-based encryption scheme remained open for many years. In 2001, Boneh and Franklin [30] proposed the first practical identity-based encryption scheme using bilinear pairings.

Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ be a Type 1 pairing, with $r = |\mathbb{G}_1| = |\mathbb{G}_T|$ and $\mathbb{G}_1 = \langle P \rangle$. Let $H_1 : \{0,1\}^* \to \mathbb{G}_1 \backslash \{O\}$ be a hash function and $H_2 : \mathbb{G}_T \to \{0,1\}^\ell$ a hash function where ℓ is the length of the message to be encrypted. We denote by A Alice's email address.
To generate private and public keys, the TTP does the following:

1. randomly selects an integer $t \in [1, r-1]$ and sets t as its private key,
2. computes $T = [t]P$ and sets T as its public key,
3. computes $D_A = [t]H_1(A)$ and securely sends it to Alice, at Alice's request for its private key.

To encrypt a message $m \in \{0,1\}^\ell$ for Alice, Bob does the following:

1. privately selects a random integer $s \in [1, r-1]$,
2. computes $S = [s]P$ and $c = m \oplus H_2\left(e(H_1(A), T)\right)^s$,
3. sends (S, c) to Alice.

To decrypt, Alice does the following:

1. receives its private key from the TTP,
2. computes $m = c \oplus H_2\left(D_A, S)\right)^s$.

The decryption works because

$$e(D_A, S) = e(tH_1(A), sP) = e(H_1(A), tP)^s = e(H_1(A), T)^s.$$

An eavesdropper who wishes to recover m from (S, c) needs to solve the BDHP.

2.2.2.3 Short signatures

Digital signatures are designed for the purpose of demonstrating the authenticity of a digital message or a document. In most digital signatures schemes, two pairs of integers modulo a large integer, say, N, are obtained as the signature. Boneh, Lynn and Shacham (BLS) [31] proposed the first signature scheme in which signatures are comprised of a single integer modulo N.

Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ be a Type 1 pairing, with $r = |\mathbb{G}_1| = |\mathbb{G}_T|$ and $\mathbb{G}_1 = \langle P \rangle$. Let $H : \{0,1\}^* \to \mathbb{G}_1 \backslash \{O\}$ be a hash function. In the BLS short signature scheme, Alice

randomly selects a secret integer $a \in [1, r-1]$ as her private key, while $A = [a]P$ is her public key. Alice's signature on a message $m \in \{0,1\}^*$ is the single $\mathbb{G}_1$-element $S = [a]M$, where $M = H(m)$. Then, a verifier will have to compute $M = H(m)$ and check that $e(P,S) = e(A,M)$. An attacker who wishes to forge Alice's signature on a message needs to solve the Diffie-Hellman problem: given $P, [a]P, [b]P$, compute $[ab]P$.

2.3 Coppersmith's index-calculus algorithm

The fastest general-purpose algorithm for solving the DLP in finite fields is the *index-calculus* method. The basic ideas of this method appeared in the early work of Kraitchik in 1922 [92], and have been rediscovered by Western and Miller in 1968 [123] (see also [104] and [102]). The refined index-calculus algorithm was independently introduced by Adleman [8], Merkle [101] and Pollard [109]. Adleman analyzed the algorithm for the case of finite fields $\mathbb{F}_q$ where q is a prime number. Hellman and Reyneri [71] extended it to fields $\mathbb{F}_q$, where q is a power of a prime number. The resulting running time is usually expressed as

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

with $0 < \alpha < 1$ and $c > 0$. This is called a *subexponential* running time.

Considerable improvements on the index-calculus algorithm have been presented by Blake, Fuji-Hara, Mullin and Vanstone [28], although these improvements do not affect the asymptotic running time of Adleman's algorithm. In 1984, Coppersmith developed a variant of the index-calculus method in fields of the form $\mathbb{F}_{2^n}$. Beside its practical efficiency, illustrated by the computation of discrete logarithms in $\mathbb{F}_{2^{127}}$, Coppersmith's algorithm improved the asymptotic running time of Adleman's algorithm by exploiting the concept of *systematic equations*, first presented by Blake et al. [28].

Let the field $\mathbb{F}_{2^n}$ be represented as $\mathbb{F}_2[X]/(X^n + h_0(X))$, where $h_0(X) \in \mathbb{F}_2[X]$ is a polynomial of 'small' degree such that $f(X) = X^n + h_0(X)$ is a *primitive polynomial*, which means that $f(X)$ is irreducible and the equivalence class of X in $\mathbb{F}_2[X]/(f(X))$ is a primitive element of the multiplicative group $\mathbb{F}^*_{2^n}$. Note that

$$X^n \equiv h_0(X) \pmod{f(X)}.$$

Then, the elements of $\mathbb{F}_{2^n}$ can be represented as polynomials in $\mathbb{F}_2[X]$ of degree at most $n-1$, and X is a generator of $\mathbb{F}^*_{2^n}$. Let $h(X) \in \mathbb{F}^*_{2^n}$, whose logarithm $\log_X h$ we wish to compute. We shall assume that $h(X)$ has degree $n-1$.

As in the original index-calculus algorithm, one selects a smoothness bound B and finds the logarithms of all irreducible polynomials of degree at most B. We define the *factor base* to be the set of all monic irreducible polynomials in $\mathbb{F}_2[X]$ of degree at most B, and notice that it has size roughly $2^{B+1}/B$. To compute the logarithms of the factor base elements, one generates linear equations relating these logarithms and solves the resulting linear system (see §2.3.1). Then, the *descent stage* (see §2.3.2) computes $\log_X h$ by expressing it as a linear combination of logarithms of the factor base elements, which are already known.

2.3.1 Computing the logarithms of the base factor elements

In Coppersmith's algorithm, one chooses $B \approx cn^{1/3}(\log n)^{2/3}$, with c a small positive constant, and selects a parameter $d \approx B$. Let k and h be positive integers such that $2^k \approx \sqrt{n/d}$ and $h = \lceil n/2^k \rceil$. To generate a linear equation, one searches for polynomials $A(X)$ and $B(X)$, with $\gcd(A(X), B(X)) = 1$, such that the polynomials $C(X)$ and $D(X)$ defined by the systematic equations

$$C(X) = X^h A(X) + B(X) \tag{2.6}$$

$$D(X) = C(X)^{2^k} \bmod f(X) \tag{2.7}$$

are both B-smooth, that is, $C(X) = \prod_i F_i(X)^{e_i}$ and $D(X) = \prod_i F_i(X)^{f_i}$ where the $F_i(X)$ are of degree at most B. In this case, taking logarithms of both sides of (2.7) gives a linear equation

$$\sum_i (f_i - ke_i) \log_X F_i \equiv 0 \pmod{2^n - 1}$$

relating logarithms of elements in the factor base. Note that the number of polynomials $A(X), B(X)$ of degree d such that $\gcd(A(X), B(X)) = 1$ is about 2^{2d+1}. After slightly more than $2^{B+1}/B$ linear equations are obtained, one solves the resulting sparse system to retrieve the logarithms of all the factor base elements.

The selections of k and h are done in such a way that $C(X)$ and $D(X)$ are both of degree about $\sqrt{nd}$. Indeed, $C(X)$ is clearly of degree at most $h + d$, and for $D(X)$ we have

$$\begin{aligned} D(X) &= \left(X^h A(X) + B(X)\right)^{2^k} \bmod f(X) \\ &= X^{h2^k} A(X)^{2^k} + B(X)^{2^k} \bmod f(X) \\ &= X^n X^{h2^k - n} A(X)^{2^k} + B(X)^{2^k} \bmod f(X) \\ &= h_0(X) X^{h2^k - n} A(X)^{2^k} + B(X)^{2^k} \bmod f(X). \end{aligned}$$

Since $h_0(X)$ has small degree and $h2^k - n \leq 2^k$, we may assume that $h_0(X)x^{h2^k-n}$ is of degree about 2^k. Therefore, $D(X)$ has degree about $(d+1)2^k$. One can easily see that both $h + d$ and $(d+1)2^k$ are about $\sqrt{nd}$.

Making the heuristic assumption that $C(X)$ and $D(X)$ behave as random polynomials of degree $\sqrt{nd}$, the probability for these polynomials to be simultaneously B-smooth is roughly

$$\left(\frac{\sqrt{nd}}{B}\right)^{-2\frac{\sqrt{nd}}{B}}$$

(more precise functions computing smoothness probabilities in practical cases are provided in §2.4). The selected range for B and d yields a probability of $1/L_{2^n}[\frac{1}{3}, \frac{2}{3\sqrt{c}}]$. Thus, the number of trials one expects in order to obtain one linear relation is $L_{2^n}[\frac{1}{3}, \frac{2}{3\sqrt{c}}]$. Since the number of required relations is (slightly more than) $2^{B+1}/B$, the total cost of the relation generation is

$$L_{2^n}\left[\frac{1}{3}, c' + o(1)\right], \quad \text{with } c' = c\log 2 + \frac{2}{3\sqrt{c}}.$$

In order to ensure that the $2^{2d+1} = L_{2^n}[\frac{1}{3}, 2c + o(1)]$ polynomials $A(X), B(X)$ are enough to provide the $2^{B+1}/B$ linear equations, one must have

$$L_{2^n}\left[\frac{1}{3}, 2c\log 2 + o(1)\right] \approx L_{2^n}\left[\frac{1}{3}, c\log 2 + \frac{2}{3\sqrt{c}} + o(1)\right],$$

and consequently

$$c \approx \left(\frac{2}{3\log 2}\right)^{2/3}.$$

Using sparse linear algebra methods such as Lanczos's or Wiedemann's algorithms, the system of linear equation can be solved at a cost of $O(\omega(2^{B+1}/B)^2)$ multiplications modulo a large prime factor of $2^n - 1$, where $\omega < \sqrt{nd}$ is the average number of nonzero terms per equation. Thus, the cost of the linear algebra solving is upper bounded by $L_{2^n}[\frac{1}{3}, c\log 2 + o(1)]$. This clearly shows that the linear algebra phase is not the bottleneck of the computation.

2.3.2 Descent stage

To express $\log_X h$ as a linear combination of logarithms of elements in the factor base, the Waterloo descent improvement [28], which we call *continued-fraction* descent in §3.3.4 and §4.2.4, is used first. In this descent phase, one repeatedly picks a random integer $m \in [1, 2^n - 1]$, runs the extended Euclidean algorithm to have $h(X)X^m \mod f(X) = w_1/w_2$, with w_1 and w_2 of degree roughly $n/2$, and then tests w_1 and w_2 for B_1-smoothness, where $B_1 \approx \sqrt{nB}$. Once B_1-smooth polynomials w_1 and w_2 are found, we obtain an expression of $\log_X h$ in terms of logarithms of irreducible polynomials with degree at most B_1. The probability for w_1, w_2 being simultaneously B_1-smooth is roughly

$$\left(\frac{n/2}{\sqrt{nB}}\right)^{-2\frac{n/2}{\sqrt{nB}}} = \frac{1}{L_{2^n}\left[\frac{1}{3}, \frac{2}{3\sqrt{c}} + o(1)\right]}.$$

Thus, the expected number of smoothness testings in this phase is approximately

$$L_{2^n}\left[\frac{1}{3}, \frac{2}{3\sqrt{c}} + o(1)\right].$$

The number of irreducible factors of w_1 and w_2 of degree between B and B_1 is at most n/B. The logarithms of these polynomials are thereafter expressed as linear combinations of logarithms of smaller degree polynomials using the following descent method.

Let $Q \in \mathbb{F}_2[X]$ be a polynomial of degree D, with $B < D \leq B_1$, and select

$$d \approx \frac{D + \sqrt{n/B}\log(n/B)}{2}.$$

Let k and h be positive integers such that $2^k \approx \sqrt{n/d}$ and $h = \lceil n/2^k \rceil$. The present method closely follows the ideas introduced in §2.3.1. Here, one sets $B' \approx \sqrt{BD}$ and searches for polynomials $A(X)$ and $B(X)$, with $\gcd(A(X), B(X)) = 1$, such that

(i) $Q(X) \mid C(X) = X^h A(X) + B(X)$;

(ii) $C(X)/Q(X)$ and $D(X) = C(X)^{2^k} \bmod f(X)$ are B'-smooth.

A family of polynomials $A(X), B(X)$ satisfying (i) can be easily constructed by finding, via the extended Euclidean algorithm, a basis of the lattice

$$L_Q = \left\{(A, B) \in \mathbb{F}_2[X] \times \mathbb{F}_2[X] \ : \ Q \mid \left(A(X)X^h - B(X)\right)\right\}.$$

When both (i) and (ii) are satisfied for some polynomials $A(X)$ and $B(X)$, then taking logarithms of both sides of the equation $D(X) = C(X)^{2^k} \bmod f(X)$ gives an expression for $\log_X Q$ in terms of logarithms of polynomials with degree at most B'. As in the first stage (§2.3.1), one can easily see that $C(X)/Q(X)$ and $D(X)$ both have degree about $\sqrt{nd}$. Then, lower bounding B' by B, one finds that the expected number of trials before obtaining B'-smooth polynomials $C(X)/Q(X)$ and $D(X)$ is upper bounded by $L_{2^n}[\frac{1}{3}, \frac{1}{3\sqrt{6}c^{5/4}} + o(1)]$.

To see that the number of polynomials $A(X)$, $B(X)$ such that $\gcd(A(X), B(X)) = 1$ is enough to ensure the expected number of trials, one must consider the minimal case where $D = B + 1$, and obtains $2^{2d+1} \approx L_{2^n}[\frac{1}{3}, \frac{\log 2}{2}(c + \frac{2}{3\sqrt{c}}) + o(1)]$. This number is in fact larger than the required the number of trials. Note that the expected number of distinct irreducible factors of $C(X)/Q(X)$ and $D(X)$ of degree between B and B' is at most $\sqrt{nd}/B < n/B$.

Now continuing the descent stage, one applies the above descent method to each of the polynomials of degree between B and B_1 coming from the first phase. Each of these polynomials requires fewer than $L_{2^n}\left[\frac{1}{3}, \frac{1}{9c^{5/4}} + o(1)\right]$ smoothness testings to be expressed as a linear combination of logarithms of fewer than n/B polynomials with degree at most $B_2 = \sqrt{BB_1} = B\,(n/B)^{1/4}$. Notice that at this point we have expressed $\log_X h$ in terms of at most $(n/B)^2$ polynomials of degree at most $B\,(n/B)^{1/2^2}$ at a cost of less than

$$L_{2^n}\left[\frac{1}{3}, \frac{2}{3\sqrt{c}} + o(1)\right] + \frac{n}{B} \cdot L_{2^n}\left[\frac{1}{3}, \frac{1}{3\sqrt{6}c^{5/4}} + o(1)\right]$$

smoothness testings. Applying the latter descent method recursively to any resulting polynomial of degree greater than B, yields after $\log n$ steps a linear relationship between $\log_X h$ and logarithms of polynomials in the factor base. This provide us $\log_X h$. The total number of smoothness testing in the descent stage is expected (for $c \geq (1/2\sqrt{6})^{4/3}$) to be less than

$$L_{2^n}\left[\frac{1}{3}, \frac{2}{3\sqrt{c}} + o(1)\right].$$

Therefore, the total running time of Coppersmith's algorithm is that of performing

$$L_{2^n}\left[\frac{1}{3}, \left(\frac{32\log 2}{9}\right)^{1/3} + o(1)\right]$$

smoothness testings. (§2.5 provides efficient ways to perform smoothness testings.)

2.4 Counting smooth polynomials

Let $\mathbb{F}_q$ be a finite field. The number of monic polynomials of degree n over $\mathbb{F}_q$ is q^n. The number of monic irreducible polynomials of degree n over $\mathbb{F}_q$ is given by Gauss's formula [54]

$$I_q(n) = \frac{1}{n}\sum_{d|n}\mu(n/d)q^d = \frac{q^n}{n} + O\left(q^{n/2}\right), \tag{2.8}$$

where $\mu : \mathbb{N} \to \mathbb{N}$ is the Möbius function

$$\mu(n) = \begin{cases} 1 & \text{if } n = 1, \\ (-1)^d & \text{if } n \text{ is product of } d \text{ distinct prime numbers.} \\ 0 & \text{otherwise.} \end{cases} \tag{2.9}$$

Let m be a nonzero positive integer. A polynomial in $\mathbb{F}_q[X]$ is said to be m-smooth if all its irreducible factors in $\mathbb{F}_q[X]$ have degree at most m. Define

$$F(u,z) = \prod_{\ell=1}^{m}\left(1 + uz^{\ell} + uz^{2\ell} + \dots\right)^{I_q(\ell)} = \prod_{\ell=1}^{m}\left(1 + \frac{uz^{\ell}}{1-z^{\ell}}\right)^{I_q(\ell)}.$$

$F(u,z)$ is the generating function for m-smooth monic polynomials in $\mathbb{F}_q[X]$, where u marks the number of distinct irreducible factors, and z marks the degree of the polynomial. Thus, the number of monic m-smooth degree-n polynomials in $\mathbb{F}_q[X]$ that have exactly k distinct monic irreducible factors is

$$N_q(m,n,k) = [u^k\, z^n]F(u,z) \tag{2.10}$$

where $[\cdot]$ denotes the coefficient operator, whereas the total number of monic m-smooth degree-n polynomials in $\mathbb{F}_q[X]$ is

$$N_q(m,n) = [z^n]F(1,z). \tag{2.11}$$

Furthermore, the average number of distinct monic irreducible factors among all monic m-smooth degree-n polynomials in $\mathbb{F}_q[X]$ is

$$A_q(m,n) = \frac{[z^n]\left(\frac{\partial F}{\partial u}\big|_{u=1}\right)}{N_q(m,n)}. \tag{2.12}$$

For any given q, m and n, $N_q(m,n)$ can be obtained by using a symbolic algebra package such as Maple [99] to compute the first $n+1$ terms of the Taylor series expansion of $F(1,z)$ and then extracting the coefficient of z^n. Similarly, one can compute $N_q(m,n,k)$ and $A_q(m,n)$. For example, we used Maple 17 on a 3.2 GHz Intel Xeon CPU X5672 machine to compute $N_{3^{12}}(30,254)$ in 3.2 seconds, $A_{3^{12}}(30,254) = 14.963$ in 102.9 seconds, and $N_{3^{12}}(30,254,9)$ in 4305 seconds.

t	Exactly t distinct irred. factors	At most t distinct irred. factors
1	0.00000000000000000000000000000000	0.00000000000000000000000000000000
2	0.00000000000000000000000000000000	0.00000000000000000000000000000000
3	0.00000000000000000000000000000000	0.00000000000000000000000000000000
4	0.00000000000000000000000000000000	0.00000000000000000000000000000000
5	0.00694396258725365727783584 2266	0.00694396258725365727783584 2266
6	0.09450823523737471257706302 1493	0.10145219782462836985489886 3759
7	0.25239461704744106489033780 3216	0.35384681487206943474523666 6975
8	0.30099673202313362746357465 5796	0.65484354689520306220881132 2771
9	0.20850598472151827911125834 8913	0.86334953161672134132006967 1684
10	0.09566691512298691622545503 3585	0.95901644673970825754552470 5269
11	0.03143115350072971206911975 9222	0.99044760024043796961464446 4492
12	0.00778172966693396340229811 1588	0.99822932990737193301694257 6080
13	0.00150455397702231797030251 0303	0.99973388388439425098724508 6382
14	0.00023317550477421944799744 9230	0.99996705938916847043524253 5613
15	0.00002953983934862953106205 0590	0.99999659922851709996630458 6203
16	0.00000310502611457732272818 3235	0.99999970425463167728903276 9438
17	0.00000027391385389124067914 1964	0.99999997816848556852971191 1402
18	0.00000002045574507926935020 3882	0.99999999862423064779906211 5284
19	0.00000000130146987379573081 8942	0.99999999992570052159479293 4226
20	0.00000000007085333588538066 1976	0.99999999999655385748017359 6201
21	0.00000000000330881436907745 8356	0.99999999999986267184925105 4557
22	0.00000000000013263322345658 5201	0.99999999999999530507270763 9758
23	0.00000000000000455771538255 9271	0.99999999999999986278809019 9029
24	0.00000000000000013380457836 3524	0.99999999999999996592668562 553
25	0.00000000000000003336091284 541	0.99999999999999999928759847 094
26	0.00000000000000000070002399 226	0.99999999999999999998762246 320
27	0.00000000000000000001220209 967	0.99999999999999999999982456 287
28	0.00000000000000000000017346 134	0.99999999999999999999999802 422
29	0.00000000000000000000000195 878	0.99999999999999999999999998 300
30	0.00000000000000000000000001 690	0.99999999999999999999999999 990
31	0.00000000000000000000000000 010	1.00000000000000000000000000 0000
32	0.00000000000000000000000000 0000	1.00000000000000000000000000 0000
33	0.00000000000000000000000000 0000	1.00000000000000000000000000 0000

Table 2.1: Proportion of monic 7-smooth degree-33 polynomials in $\mathbb{F}_{3^{12}}[X]$ that have exactly t (resp. at most t) distinct monic irreducible factors.

Example 2.30. ($q = 3^{12}$, $n = 33$, $m = 7$) Table 2.1 lists, for each $t \in [1, 33]$, the proportion of monic 7-smooth degree-33 polynomials in $\mathbb{F}_{3^{12}}[X]$ that have exactly t distinct monic irreducible factors, and the proportion that have at most t distinct monic irreducible factors (cf. §3.4.6). We see that most 7-smooth degree-33 polynomials in $\mathbb{F}_{3^{12}}[X]$ will have 6, 7, 8, 9 or 10 distinct monic irreducible factors. In fact, the average number of distinct monic irreducible factors is $A_{3^{12}}(7, 33) = 8.072$.

2.5 Smoothness testing

Theorem 2.31 (Smoothness test [40])**.** Let $\mathbb{F}_q$ be a finite field of characteristic p and f a monic polynomial in $\mathbb{F}_q[X]$ of degree d, $d > 0$. For an integer m, $1 \leq m \leq d$, define

$$w(X) = f'(X) \cdot \prod_{i=\lceil m/2 \rceil}^{m} (X^{q^i} - X) \mod f(X). \tag{2.13}$$

If f is m-smooth, then $w = 0$; if f is not m-smooth, then $w = 0$ if and only if p divides the multiplicity of all irreducible factors of f with degree greater than m.

Proof. It is known that for any positive integer i, the polynomial $X^{q^i} - X$ is the product of all the monic irreducible polynomials in $\mathbb{F}_q[X]$ whose degree divides i [96, Theorem 3.20]. Thus, the polynomial $P = \prod_{i=\lceil m/2 \rceil}^{m} (X^{q^i} - X)$ is the product of all the monic irreducible polynomials of degree at most m. Since P never vanishes, we have $w = 0$ if and only if $f' = 0$ or $f | f'P$. One can see that $f' = 0$ if and only f is a polynomial in X^p; and when f is not a polynomial in X^p, the multiplicity in f' of an irreducible factor of f with multiplicity $e \geq 1$ in f is either e if e is a multiple of p or $e - 1$ otherwise. Therefore, $w = 0$ corresponds to f being a polynomial in X^p or $f | f'P$. Now suppose $f | f'P$. This means that all the irreducible factors of f with multiplicity not a multiple of p are of degree at most m, because, for these factors, the multiplicity gap of 1 arising from the derivation of f could only have been filled during the multiplication by P, whose irreducible factors are all of degree at most m. Hence, if f is not m-smooth, $w = 0$ happens if and only if all the irreducible factors of degree greater than m are of multiplicity a multiple of p, since the latter statement is also realized when f is a polynomial in X^p. Conversely, it is clear that when f is m-smooth we necessarily have $w = 0$. □

For a given positive integer m, since a randomly selected polynomial is unlikely to satisfy the condition of having all its irreducible factors of degree greater than m coming with multiplicity divisible by the characteristic of the finite field, the vast majority of polynomials that pass the smoothness test are indeed m-smooth. The polynomials that are declared to be m-smooth are then factored using a general-purpose polynomial factorization algorithm, at which time the very few polynomials falsely declared to be m-smooth are identified.

2.5.1 Basic method for evaluating $w(X)$

Let $f(X)$ be a polynomial over a finite field $\mathbb{F}_q$. Without loss of generality, we can assume that f is monic. Then the product of two polynomials of degree $< d$ can be multiplied

modulo f in time $2d^2$, where the unit of time is an $\mathbb{F}_q$-multiplication. To compute $w(X) = f' \prod_{i=\lceil m/2 \rceil}^{m} (X^{q^i} - X) \bmod f$, one first precomputes $X^q \bmod f$. This can be accomplished by repeated square-and-multiplication at a cost of at most $2||q||_2$ modular multiplications, where $||q||_2$ denotes the bitlength of q. Then, $X^{q^i} \bmod f$ for $2 \leq i \leq d-1$ can be computed by repeated multiplication of $X^q \bmod f$ with itself at a cost of approximately d modular multiplications, and $X^{q^i} \bmod f$ for $2 \leq i \leq m$ can be computed by repeated exponentiation by q with each exponentiating having cost d^2 $\mathbb{F}_q$-multiplications. Finally, the product in (2.13) can be computed using $m/2$ modular multiplications at a cost of md^2 $\mathbb{F}_q$-multiplications. The total cost for testing m-smoothness of f is thus

$$S_q(m, d) = 2d^2(d + m + 2||q||_2) \quad \mathbb{F}_q\text{-multiplications.} \tag{2.14}$$

This method is used in Chapters 3 and 4. For example, consider the case where $q = 3^{12}$. Then, $X^q \bmod f$ can be determined by first precomputing $X^3, X^6, \ldots, X^{3(d-1)} \bmod f$ by repeated multiplication by X. Thereafter, cubing a polynomial modulo f can be accomplished by cubing the coefficients of the polynomial, and then multiplying the precomputed polynomials by these cubes (and adding the results). In this way, we get a loose upper bound of $3d^2 + 11d^2 = 14d^2$ $\mathbb{F}_{3^{12}}$-multiplications of the cost to compute $X^{3^{12}} \bmod f$, and the total cost for testing m-smoothness of f becomes

$$S_{3^{12}}(m, d) = 2d^2(d + m + 7) \quad \mathbb{F}_{3^{12}}\text{-multiplications.} \tag{2.15}$$

2.5.2 Improved method for evaluating $w(X)$

In [59], Granger et al. presented some improvements on performing smoothness testings over polynomials defined over finite fields of order a power of a prime.

Let $q = p^\ell$, where p is a small prime number and ℓ a positive composite integer expressed as $\ell = r \cdot s$. Let $f \in \mathbb{F}_q[X]$ be a polynomial of degree n that we want to test for m-smoothness ($1 \leq m \leq n$) and consider the quotient ring $\mathbb{F}_q[X]/(f)$ where the class of a polynomial $g(X)$ over $\mathbb{F}_q$ is written $[g(X)]$. For the sake of simplicity, without loss of generality, it will be assumed throughout the following analysis that the cost of elementary field arithmetic operations such as addition, multiplication and exponentiation have all the same computational cost. In the case of the ring arithmetic, it will be further assumed that the cost of a polynomial addition is $O(n)$, whereas the polynomial ring multiplication or division has a cost of $O(n^2)$.

The approach of Granger *et al.* can be described as follows. One first computes $[X^{p^{rs}}]$ followed by the exponentiations $[X^{q^i}]$, for $i = 1, \ldots, m$, according to the steps described hereafter.

- **Precomputations: For $i = 1, \ldots, n-1$, calculate $X^{ip^r} \bmod f$.** The exponentiations $X^{p^r}, X^{2p^r}, \ldots, X^{(n-1)p^r}$ can be computed by consecutively multiplying by X. In general, a multiplication by X can be achieved by a shifting and possibly followed by a reduction modulo f. Notice that a polynomial having the same degree as f can be reduced modulo f at a computational cost of m field multiplications and additions in $\mathbb{F}_q$. Hence, the precomputation phase can be accomplished with $(n-1)(p^r - 1)$ shift operations at a computational cost less than $2n^2p^r$ field operations in $\mathbb{F}_q$.

- **Performing a ring exponentiation** a^{p^r}. Using the $n-1$ precomputed elements of the first step, one can perform the exponentiation a^{p^r} for any arbitrary $a \in \mathbb{F}_q[X]/(f)$ as

$$a^{p^r} = \left[\sum_{i=0}^{n-1} a_i X^i\right]^{p^r} = \sum_{i=0}^{n-1} a_i^{p^r} \left[X^{ip^r}\right]. \tag{2.16}$$

  This computation requires n exponentiations and n scalar multiplications (that is, the operation of multiplying a field element by a degree-$(n-1)$ polynomial). Hence, the total cost of this operation is about $2n^2 + n$ field operations in $\mathbb{F}_q$.

- **Computing** $[X^{q^i}]$, **for** $i = 1, \ldots, m$**.** Recall that $q = p^{rs}$. Hence, applying repeatedly the second step, one can compute $[X^{p^{rj}}]$ for $j \in \{2, \ldots, sm\}$, and therefore obtains $[X^{q^i}] = [X^{p^{rsi}}]$ for $i \in \{1, \ldots, m\}$. The cost of this step is of approximately $(2n^2 + n)sm = 2n^2sm + nsm$ operations in $\mathbb{F}_q$.

The cost of multiplying or dividing two degree-$(n-1)$ polynomials is about n^2 operations over $\mathbb{F}_q$. Hence, a quotient ring multiplication, that is, the cost of multiplying two degree-$(n-1)$ polynomials modulo f, has a cost of roughly $2n^2$ field operations. The computation of $w(X) = f' \prod_{i=\lceil m/2 \rceil}^{m} (X^{q^i} - X) \bmod f$ additionally requires performing $\lceil m/2 \rceil$ quotient ring multiplications, with total cost $n^2 m$ arithmetic operations in $\mathbb{F}_q$.

Summarizing, the cost of computing $w(X)$ can be estimated as

$$S_q(m, d) = 2n^2p^r + 2n^2sm + nsm + n^2m \quad \mathbb{F}_q\text{-multiplications.} \tag{2.17}$$

3 Weakness of $\mathbb{F}_{3^{6\cdot509}}$ and $\mathbb{F}_{2^{4\cdot3041}}$ for Discrete Logarithm Cryptography

3.1 Introduction

We shall assume in this chapter that the characteristic of all finite fields is 2 or 3. With this setting, the fastest general-purpose algorithm known for solving the DLP in a field $\mathbb{F}_Q$ is Coppersmith's 1984 index-calculus algorithm [40] with a running time of $L_Q[\frac{1}{3}, (32/9)^{1/3}] \approx L_Q[\frac{1}{3}, 1.526]$, where as usual $L_Q[\alpha, c]$ with $0 < \alpha < 1$ and $c > 0$ denotes the expression

$$\exp\left((c + o(1))(\log Q)^{\alpha}(\log\log Q)^{1-\alpha}\right).$$

However, when the parameters q and n of the finite field $\mathbb{F}_{q^n}$ are balanced in some way, we may have faster algorithms. Hence, when q and n are balanced in the sense that

$$q = \exp\left(3^{-2/3} \cdot (\log Q)^{1/3}(\log\log Q)^{2/3}\right) \text{ and } n = 3^{2/3} \cdot \left(\frac{\log Q}{\log\log Q}\right)^{2/3},$$

the Joux-Lercier (2006) algorithm [81], performs better than Coppersmith's algorithm, with a running time of $L_Q[\frac{1}{3}, 3^{1/3}] \approx L_Q[\frac{1}{3}, 1.442]$. In 2012, Joux [76] introduced a 'pinpointing' technique that improved the running time of the Joux-Lercier algorithm to $L_Q[\frac{1}{3}, 2/3^{2/3}] \approx L_Q[\frac{1}{3}, 0.961]$.

In February 2013, Joux [77] presented a new index-calculus algorithm for solving the DLP with a running time of $L_Q[\frac{1}{4} + o(1), c]$ (for some undetermined c) when q and n are balanced in the sense that $q \approx m$ where $n = 2m$. Also in February 2013, Göloğlu, Granger, McGuire and Zumbrägel [57] proposed a variant of the Joux-Lercier algorithm that imposes a further divisibility condition on ℓ where $q = 2^{\ell}$. The running time of the Gögloğlu et al. algorithm is (i) $L_Q[\frac{1}{3}, 2/3^{2/3}] \approx L_Q[\frac{1}{3}, 0.961]$ when $n \approx 2^m d_1$, $d_1 \approx 2^m$, and $m \mid \ell$; and (ii) between $L_Q[\frac{1}{3}, (2/3)^{2/3}] \approx L_Q[\frac{1}{3}, 0.763]$ and $L_Q[\frac{1}{3}, 1/2^{1/3}] \approx L_Q[\frac{1}{3}, 0.794]$ when $n \approx 2^m d_1$, $2^m \gg d_1$, and $m \mid \ell$. The new algorithms were used to compute discrete logarithms in $\mathbb{F}_{2^{8\cdot3\cdot255}} = \mathbb{F}_{2^{6120}}$ in only 750 CPU hours [58], and in $\mathbb{F}_{2^{8\cdot3\cdot257}} = \mathbb{F}_{2^{6168}}$ in only 550 CPU hours [78]. The astoundingly small computational effort expended in these experiments depends crucially on the special nature of the fields $\mathbb{F}_{2^{6120}}$ and $\mathbb{F}_{2^{6168}}$ – namely that $\mathbb{F}_{2^{6120}}$ is a degree-255 extension of $\mathbb{F}_{2^{8\cdot3}}$ with $255 = 2^8 - 1$, and $\mathbb{F}_{2^{6168}}$ is a degree-257 extension of $\mathbb{F}_{2^{8\cdot3}}$ with $257 = 2^8 + 1$. Despite these remarkable achievements, the effectiveness of the new algorithms for computing discrete logarithms in general finite fields of small characteristic remained unclear.

In June 2013, Barbulescu, Gaudry, Joux and Thomé [16] presented a new DLP algorithm that, for many choices of field sizes, is asymptotically faster than all previous algorithms.

Most impressively, in the case where $q \approx n$ and $n \leq q+2$, the discrete logarithm problem in $\mathbb{F}_{q^{2n}} = \mathbb{F}_Q$ can be solved in *quasi-polynomial time*

$$(\log Q)^{O(\log \log Q)}. \tag{3.1}$$

Note that (3.1) is asymptotically smaller than $L_Q[\alpha, c]$ for any $\alpha > 0$ and $c > 0$. However, the practical relevance of the new algorithm has not yet been determined.

The aforementioned advances in DLP algorithms are potentially relevant to the security of Type 1 pairing-based cryptosystems that use bilinear pairings derived from supersingular elliptic curves or genus-2 hyperelliptic curves defined over finite fields $\mathbb{F}_q$ of characteristic 2 or 3. Three such symmetric pairings that have received a great deal of attention in the literature are:

- The $k = 6$ pairings derived from supersingular elliptic curves $y^2 = x^3 - x + 1$ and $y^2 = x^3 - x - 1$ over $\mathbb{F}_{3^\ell}$;
- The $k = 4$ pairings derived from supersingular elliptic curves $y^2 + y = x^3 + x$ and $y^2 + y = x^3 + x + 1$ over $\mathbb{F}_{2^\ell}$;
- The $k = 12$ pairing derived from supersingular genus-2 curves $y^2 + y = x^5 + x^3$ and $y^2 + y = x^5 + x^3 + 1$ over $\mathbb{F}_{2^\ell}$,

where, in all cases, ℓ is chosen to be prime and k is the embedding degree of the pairing. These symmetric pairings were considered in some early papers [31, 51, 18, 52] on pairing-based cryptography. Since then, many papers have reported on software and hardware implementation of these pairings; some examples are [17, 62, 108, 9, 67, 27, 35, 46, 25, 10, 2].

In all the papers cited in the previous paragraph, the pairing parameters were chosen under the assumption that Coppersmith's algorithm is the fastest method for finding discrete logarithms in $\mathbb{F}_{p^{k\cdot\ell}}$, $p = 2$ or 3. For example, to achieve the 128-bit security level, [9] chose $\ell = 1223$ for the $k = 4$ pairing and $\ell = 509$ for the $k = 6$ pairing, [35] chose $\ell = 439$ for the $k = 12$ pairing, and [10] chose $\ell = 367$ for the $k = 12$ pairing. These choices were made because Coppersmith's algorithm, as analyzed by Lenstra [94], has running time approximately 2^{128} for computing logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$, $\mathbb{F}_{3^{6\cdot 509}}$, $\mathbb{F}_{2^{12\cdot 439}}$, and $\mathbb{F}_{2^{12\cdot 367}}$, respectively.

In 2012, Hayashi et al. [69] reported on their implementation of the Joux-Lercier algorithm for computing logarithms in $\mathbb{F}_{3^{6\cdot 97}}$. Their work demonstrated that in practice the Joux-Lercier algorithm is considerably faster than Coppersmith's algorithm for DLP computations in $\mathbb{F}_{3^{6\cdot 97}}$; note that the $k = 6$ pairing with $\ell = 97$ was considered in [18, 52]. In contrast, the largest discrete logarithm computation reported using Coppersmith's algorithm (and its generalizations [7, 79]) is the April 2013 computation by Barbulescu et al. [14] of logarithms in $\mathbb{F}_{2^{809}}$; note that 809 is prime and $3^{6\cdot 97} \approx 2^{922}$. Shinohara et al. [115] estimated that $\mathbb{F}_{3^{6\cdot 509}}$ offers only 111-bits of security against Joux-Lercier attacks, considerably less than the assumed 128-bits of security against Coppersmith attacks.

The purpose of the analysis presented in this chapter, in joint work with A. Menezes, T. Oliveira and F. Rodríguez-Henríquez [3, 4], is to demonstrate that the new algorithms by Joux [77] and Barbulescu et al. [16] can be combined to solve the discrete logarithm problem

in $\mathbb{F}_{3^{6\cdot 509}}$ and $\mathbb{F}_{2^{12\cdot 367}}$ significantly faster than the Joux-Lercier algorithm. More precisely, we estimate that logarithms in these fields can be computed in $2^{81.7}$ and 2^{100} time, respectively, with the new algorithms, where the unit of time is the (inexpensive) cost of a multiplication in $\mathbb{F}_{3^{12}}$ and $\mathbb{F}_{2^{24}}$, respectively.

In addition, we show that the new algorithms can have a drastic impact on the security of the supersingular elliptic curves of embedding degree 4 at higher levels of security. More precisely, we consider the embedding degree-4 elliptic curve $E : y^2 + y = x^3 + x$ over $\mathbb{F}_{2^{3041}}$, where $|E(\mathbb{F}_{2^{3041}})| = r$ with r a 3041-bit prime. The finite field $\mathbb{F}_{2^{4\cdot 3041}}$ offers approximately 192 bits of security against attacks on the DLP by Coppersmith's algorithm. In contrast, our concrete analysis shows that the order-r subgroup of the multiplicative group of this field offers at most 129 bits of security against the new attacks.

Moreover, the computations are effectively parallelizable, whereas the Joux-Lercier algorithm and Coppersmith's algorithm are not because of the very large size of the linear system of equations that needs to be solved. While the $2^{81.7}$ computation is certainly a formidable challenge, it is already within the realm of feasibility for a very well-funded adversary. Thus, we conclude that $\mathbb{F}_{3^{6\cdot 509}}$ does not offer adequate security for discrete logarithm cryptosystems and, in particular, the supersingular elliptic curve over $\mathbb{F}_{3^{509}}$ with embedding degree 6 is not suitable for implementing pairing-based cryptosystems. We also conclude that the supersingular genus-2 curve over $\mathbb{F}_{2^{367}}$ with embedding degree 12 and the supersingular elliptic curve over $\mathbb{F}_{2^{3041}}$ with embedding degree 4 should be considered weak and not employed in pairing-based cryptography.

The remainder of the chapter is organized as follows. In §3.2, we elaborate on the "asymptotic nature" of the Barbulescu et al. algorithm and make a case for a concrete analysis of the new DLP algorithms. In §3.3 we outline the new discrete logarithm algorithms. Our estimates for discrete logarithm computations in $\mathbb{F}_{3^{6\cdot 509}}$, $\mathbb{F}_{2^{12\cdot 367}}$ and $\mathbb{F}_{2^{4\cdot 3041}}$ are presented in §3.4, §3.5 and §3.6, respectively. We draw the conclusions of this chapter in §3.7.

3.2 On the asymptotic nature of the QPA algorithm

Let E denote the embedding degree-4 supersingular elliptic curve $y^2 + y = x^3 + x$ or $y^2 + y = x^3 + x + 1$ over $\mathbb{F}_{2^n}$ where n is prime, and suppose that $|E(\mathbb{F}_{2^n})| = cr$ where r is prime and $c \ll r$. The Weil and Tate pairings reduce the discrete logarithm problem in the order-r subgroup of $E(\mathbb{F}_{2^n})$ to the discrete logarithm problem in the order-r subgroup of the multiplicative group of $\mathbb{F}_{2^{4n}}$. Coppersmith's subexponential-time algorithm [40] can be used to solve the latter problem.

In contrast, the QPA algorithm of Barbulescu et al. [16] tackles the problem by embedding $\mathbb{F}_{2^{4n}}$ in $\mathbb{F}_{q^{2n}}$ where $q = 2^\ell \approx n$. The running time of the QPA algorithm is dominated by the descent stage. In this stage, one begins with a field element seen as a polynomial of degree (at most) $n - 1$ over $\mathbb{F}_{q^2}$ whose logarithm is sought. One then expresses the logarithm of this polynomial in terms of the logarithms of roughly q^2 polynomials of degree at most $n/2$. This process is applied recursively to each polynomial encountered in the "descent tree"; the logarithm of each such polynomial of degree d is expressed in terms of the logarithms of roughly q^2 polynomials of degree at most $d/2$. To terminate the recursion, the logarithms

of all degree-one polynomials are obtained using a relatively fast method. The number of nodes in the descent tree gives a very crude lower bound on the running time of the QPA algorithm. Since $n \approx q$, the descent tree has approximately $\log_2 q$ levels and at least $q^{2\log_2 q}$ nodes.

Table 3.1 compares the running time $C(q) = \exp(1.526(\log 2^{4q})^{1/3}(\log\log 2^{4q})^{2/3})$ of Coppersmith's algorithm for computing discrete logarithms in $\mathbb{F}_{2^{4q}}$, and the lower bound $q^{2\log_2 q}$ on the running time of the QPA algorithm for computing discrete logarithms in $\mathbb{F}_{q^{2n}}$ with $q \approx n$. We see from Table 3.1 that the QPA algorithm is faster than Coppersmith's algorithm

q	$q^{2\log_2 q}$	$C(q)$
2^9	2^{162}	2^{93}
2^{10}	2^{200}	2^{124}
2^{11}	2^{242}	2^{165}
2^{12}	2^{288}	2^{219}
2^{13}	2^{338}	2^{290}
2^{14}	2^{392}	2^{382}
2^{15}	2^{450}	2^{501}

Table 3.1: Comparison of the running time $q^{2\log_2 q}$ of the QPA algorithm for computing logarithms in $\mathbb{F}_{q^{2n}}$ with $q \approx n$, and the running time $C(q)$ of Coppersmith's algorithm for computing logarithms in $\mathbb{F}_{2^{4n}}$.

only when $n \approx q = 2^{15}$. However, such n is too large to be of interest in cryptography based on pairings over $E(\mathbb{F}_{2^n})$.

As already stated in [16, Section 6.2], to determine the practical efficiency of the QPA algorithm, and therefore the implications of QPA to the security of pairing-based cryptosystems based on $E(\mathbb{F}_{2^n})$, it is imperative that the descent stage of QPA be combined with descent steps from classical algorithms. The asymptotic running time of the resulting hybrid algorithm is difficult to determine. Instead, the framework and tools introduced in this chapter are used to perform a concrete analysis which provides a reasonably accurate picture of the effectiveness of the hybrid algorithm.

3.3 New DLP algorithm of Joux and Barbulescu et al.

The DLP algorithm we describe is due to Joux [77], with a descent step from the quasi-polynomial time algorithm (QPA) of Barbulescu et al. [16]. For lack of a better name, we will call this algorithm the "new DLP algorithm".

Let $\mathbb{F}_{q^{2n}}$ be a finite field where $n \leq q+2$. The elements of $\mathbb{F}_{q^{2n}}$ are represented as polynomials of degree at most $n-1$ over $\mathbb{F}_{q^2}$. Let $N = q^{2n}-1$. Let g be an element of order N in $\mathbb{F}^*_{q^{2n}}$, and let $h \in \mathbb{F}^*_{q^{2n}}$. We wish to compute $\log_g h$. The algorithm proceeds by first finding the logarithms of all degree-one (§3.3.2) and degree-two (§3.3.3) elements in $\mathbb{F}_{q^{2n}}$. Then, in the descent stage, $\log_g h$ is expressed as a linear combination of logarithms of degree-one and degree-two $\mathbb{F}_{q^{2n}}$ elements. The descent stage proceeds in several steps, each

expressing the logarithm of a degree-D element as a linear combination of the logarithms of elements of degree $\leq m$ for some $m < D$. Four descent methods are used; these are described in §3.3.4–§3.3.7. The cost of each step is given in Table 3.2.

Finding logarithms of linear polynomials (§3.3.2)	
Relation generation	$6q^2 \cdot S_{q^2}(1,3)$
Linear algebra	$q^5 \cdot A_N$
Finding logarithms of irreducible quadratic polynomials (§3.3.3)	
Relation generation	$q^{16}/N_{q^2}(1,6) \cdot S_{q^2}(1,6)$
Linear algebra	$q^7 \cdot A_N$
Descent (Degree D to degree m)	
Continued-fraction (§3.3.4)	$\{D = n-1\}\ \left(\frac{q^{n-1}}{N_{q^2}(m,(n-1)/2)}\right)^2 \cdot S_{q^2}(m,(n-1)/2)$
Classical (§3.3.5)	$\frac{q^{2(t_1-D+t_2)}}{N_{q^2}(m,t_1-D)N_{q^2}(m,t_2)} \cdot \min(S_{q^2}(m,t_1-D), S_{q^2}(m,t_2))$
QPA (§3.3.6)	$\frac{q^{6D+2}}{N_{q^2}(m,3D)} \cdot S_{q^2}(m,3D) + q^5 \cdot A_N$
Gröbner bases (§3.3.7)	$G_{q^2}(m,D) + \frac{q^{6m-2D}}{N_{q^2}(m,3m-D)} \cdot S_{q^2}(m,3m-D)$

Table 3.2: Estimated costs of the main steps of the new DLP algorithm for computing discrete logarithms in $\mathbb{F}_{q^{2n}}$. A_N and M_{q^2} denote the costs of an addition modulo N and a multiplication in $\mathbb{F}_{q^2}$. The smoothness testing cost $S_{q^2}(m,D)$ is given in (2.17). See §3.3.5 for the definitions of t_1 and t_2. The Gröbner basis cost $G_{q^2}(m,D)$ is defined in §3.3.7.

Notation. For $\gamma \in \mathbb{F}_{q^2}$, $\overline{\gamma}$ denotes the element γ^q. For $P \in \mathbb{F}_{q^2}[X]$, $\overline{P}$ denotes the polynomial obtained by raising each coefficient of P to the power q. The cost of an integer addition modulo N is denoted by A_N, and the cost of a multiplication in $\mathbb{F}_{q^2}$ is denoted by M_{q^2}. The projective general linear group of order 2 over $\mathbb{F}_q$ is denoted $\mathrm{PGL}_2(\mathbb{F}_q)$. $\mathcal{P}_q$ is a set of distinct representatives of the left cosets of $\mathrm{PGL}_2(\mathbb{F}_q)$ in $\mathrm{PGL}_2(\mathbb{F}_{q^2})$; note that $|\mathcal{P}_q| = q^3 + q$.

3.3.1 Setup

Select polynomials $h_0, h_1 \in \mathbb{F}_{q^2}[X]$ of degree at most 2 so that $h_1 X^q - h_0$ has an irreducible factor I_X of degree n in $\mathbb{F}_{q^2}[X]$; we will henceforth assume that $\max(\deg h_0, \deg h_1) = 2$. In order to avoid the "traps" discussed in [36], we further assume that each irreducible factor $J \in \mathbb{F}_{q^2}[X]$ of $(h_1 X^q - h_0)/I_X$ satisfies the following two conditions: (i) $\gcd(\deg J, n) = 1$; and (ii) $\deg J > m$ where m is the integer specified in the continued-fraction descent stage (§3.3.4). Note that

$$X^q \equiv h_0/h_1 \pmod{I_X}.$$

The field $\mathbb{F}_{q^{2n}}$ is represented as $\mathbb{F}_{q^{2n}} = \mathbb{F}_{q^2}[X]/(I_X)$ and the elements of $\mathbb{F}_{q^{2n}}$ can be represented as polynomials in $\mathbb{F}_{q^2}[X]$ of degree at most $n-1$. Let g be a generator of $\mathbb{F}_{q^{2n}}^*$.

3.3.2 Finding logarithms of linear polynomials

Let $\mathcal{B}_1 = \{X + a \mid a \in \mathbb{F}_{q^2}\}$, and note that $|\mathcal{B}_1| = q^2$. To compute the logarithms of $\mathcal{B}_1$-elements, we first generate linear relations of these logarithms. Let $a, b, c, d \in \mathbb{F}_{q^2}$ with $ad - bc \neq 0$. Substituting $Y \mapsto (aX + b)/(cX + d)$ into the systematic equation

$$Y^q - Y = \prod_{\alpha \in \mathbb{F}_q} (Y - \alpha), \tag{3.2}$$

and then multiplying by $h_1(cX + d)^{q+1}$ yields

$$\begin{aligned}(\overline{a}h_0 + \overline{b}h_1)(cX + d) - (aX + b)(\overline{c}h_0 + \overline{d}h_1) \\ \equiv h_1 \cdot (cX + d) \cdot \prod_{\alpha \in \mathbb{F}_q} [(a - \alpha c)X + (b - \alpha d)] \pmod{I_X}.\end{aligned} \tag{3.3}$$

Note that the left side of (3.3) is a polynomial of degree (at most) 3. If this polynomial is 1-smooth, then taking logarithms of both sides of (3.3) yields a linear relation of the logarithms of $\mathcal{B}_1$-elements[1] and the logarithm of h_1. As explained in [16], in order to avoid redundant relations one selects quadruples (a, b, c, d) from $\mathcal{P}_q$; here we are identifying a quadruple (a, b, c, d) with the matrix $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$.

Now, the probability that the left side of (3.3) is 1-smooth is

$$\frac{N_{q^2}(1, 3)}{q^6} = \binom{q^2 + 2}{3} / q^6 \approx \frac{1}{6}.$$

Thus, after approximately $6q^2$ trials one expects to obtain (slightly more than) q^2 relations. The cost of the relation generation stage is $6q^2 \cdot S_{q^2}(1, 3)$. The logarithms can then be obtained by using Wiedemann's algorithm for solving sparse systems of linear equations [124]. The expected cost of the linear algebra is $q^5 \cdot A_N$ since each equation has approximately q nonzero terms.

Remark 3.1. (*running time of Wiedemann's algorithm*) Let B be the matrix obtained after the relation generation stage. Note that B is a matrix over $\mathbb{Z}_N$. However, the entries of B are coefficients of the discrete logarithms of linear polynomials that occur in the relations. Thus the vast majority of these entries are expected to be 0, 1, and -1, with the remaining entries (corresponding to repeated factors) being a number that is small in absolute value (e.g. ± 2). Wiedemann's algorithm treats B as a black box, and uses it only to perform matrix-vector multiplication with vectors over $\mathbb{Z}_N$. Since the nonzero entries of B are very small in absolute value, and since B has approximately q nonzero entries per row, the expected cost of each matrix-by-vector multiplication is $q^3 \cdot A_N$. Finally, since the block version of Wiedemann's algorithm [41] requires no more than q^2 such matrix-by-vector multiplications, the overall running time is $q^5 \cdot A_N$.

[1] It is understood that all polynomials of the right side of (3.3) and factors of the left side of (3.3) should be made monic. The same holds for (3.10) and (3.12).

3.3.3 Finding logarithms of irreducible quadratic polynomials

Let $u \in \mathbb{F}_{q^2}$, and let $Q(X) = X^2 + uX + v \in \mathbb{F}_{q^2}[X]$ be an irreducible quadratic. Define $\mathcal{B}_{2,u}$ to be the set of all irreducible quadratics of the form $X^2 + uX + w$ in $\mathbb{F}_{q^2}[X]$; one expects that $|\mathcal{B}_{2,u}| \approx (q^2 - 1)/2$. The logarithms of all elements in $\mathcal{B}_{2,u}$ are found simultaneously using one application of QPA descent (see §3.3.6). More precisely, one first collects relations of the form (3.10), where the left side of (3.10) factors as a product of linear polynomials (whose logarithms are known). The expected number of relations one can obtain is

$$\frac{N_{q^2}(1,6)}{q^{12}} \cdot (q^3 + q).$$

Provided that this number is significantly greater than $|\mathcal{B}_{2,u}|$, the matrix $\mathcal{H}(Q)$ is expected to have full (column) rank. One can then solve the resulting system of linear equations to obtain the logarithms of all irreducible translates $Q + w$ of Q. This step is repeated for each $u \in \mathbb{F}_{q^2}$. Hence, there are q^2 independent linear systems of equations to be solved.

For each $u \in \mathbb{F}_{q^2}$, the cost of relation generation is $q^{14}/N_{q^2}(1,6) \cdot S_{q^2}(1,6)$, while the linear algebra cost is $q^5 \cdot A_N$.

3.3.4 Continued-fraction descent

Recall that we wish to compute $\log_g h$, where $h \in \mathbb{F}_{q^{2n}} = \mathbb{F}_{q^2}[X]/(I_X)$. Note that $\deg h \leq n-1$; we will henceforth assume that $\deg h = n-1$. The descent stage begins by multiplying h by a random power of g. The extended Euclidean algorithm is used to express the resulting field element h' in the form $h' = w_1/w_2$ where $\deg w_1, \deg w_2 \approx n/2$ [28]; for simplicity, we shall assume that n is odd and $\deg w_1 = \deg w_2 = (n-1)/2$. This process is repeated until both w_1 and w_2 are m-smooth for some chosen $m < (n-1)/2$. This gives $\log_g h'$ as a linear combination of logarithms of polynomials of degree at most m. The expected cost of this continued-fraction descent step is approximately

$$\left(\frac{q^{n-1}}{N_{q^2}(m,(n-1)/2)}\right)^2 \cdot S_{q^2}(m,(n-1)/2). \tag{3.4}$$

The expected number of distinct irreducible factors of w_1 and w_2 is $2A_{q^2}(m,(n-1)/2)$. In the analysis, we shall assume that each of these irreducible factors has degree exactly m. The logarithm of each of these degree-m polynomials is then expressed as a linear combination of logarithms of smaller degree polynomials using one of the descent methods described in §3.3.5, §3.3.6 and §3.3.7.

3.3.5 Classical descent

Let p be the characteristic of $\mathbb{F}_q$, and let $q = p^\ell$. Let $s \in [1,\ell]$, and let $R \in \mathbb{F}_{q^2}[X,Y]$. Then

$$R(X, X^{p^s})^{p^{\ell-s}} = R'(X^{p^{\ell-s}}, X^q) \equiv R'(X^{p^{\ell-s}}, \frac{h_0}{h_1}) \pmod{I_X},$$

where R' is obtained from R by raising all its coefficients to the power $p^{\ell-s}$. For the sake of simplicity, we will assume in this section that $h_1 = 1$ and so

$$R(X, X^{p^s})^{p^{\ell-s}} \equiv R'(X^{p^{\ell-s}}, h_0) \pmod{I_X}. \tag{3.5}$$

Let $Q \in \mathbb{F}_{q^2}[X]$ with $\deg Q = D$, and let $m < D$. In the Joux-Lercier descent method [81], as modified by Joux [77], one selects suitable parameters d_1, d_2 and searches for a polynomial $R \in \mathbb{F}_{q^2}[X, Y]$ such that (i) $\deg_X R \leq d_1$ and $\deg_Y R \leq d_2$; (ii) $Q \mid R_1$ where $R_1 = R(X, X^{p^s})$; and (iii) R_1/Q and R_2 are m-smooth where $R_2 = R'(X^{p^{\ell-s}}, h_0)$. Taking logarithms of both sides of (3.5) then gives an expression for $\log_g Q$ in terms of the logarithms of polynomials of degree at most m.

A family of polynomials R satisfying (i) and (ii) can be constructed by finding the null space of the $D \times (D + \delta)$ matrix whose columns are indexed by monomials $X^i Y^j$ for $D + \delta$ pairs $(i, j) \in [0, d_1] \times [0, d_2]$, and whose $X^i Y^j$-th column entries are the coefficients of the polynomial $X^i (X^{p^s})^j \bmod Q$. The components of the vectors in the null space of this matrix can be interpreted as the coefficients of polynomials $R \in \mathbb{F}_{q^2}[X, Y]$ satisfying (i) and (ii). The dimension of this null space is expected to be δ, and so the null space is expected to contain $(q^2)^{\delta-1}$ monic polynomials. Let $\deg R_1 = t_1$ and $\deg R_2 = t_2$. We have $t_1 \leq d_1 + p^s d_2$ and $t_2 \leq p^{\ell-s} d_1 + 2d_2$; the precise values of t_1 and t_2 depend on the (i, j) pairs chosen (see §3.4.5 for an example). In order to ensure that the null space includes a monic polynomial R such that both R_1/Q and R_2 are m-smooth, the parameters must be selected so that

$$q^{2\delta-2} \gg \frac{q^{2(t_1-D)}}{N_{q^2}(m, t_1 - D)} \cdot \frac{q^{2t_2}}{N_{q^2}(m, t_2)}. \tag{3.6}$$

Ignoring the time to compute the null space, the expected cost of finding a polynomial R satisfying (i)–(iii) is

$$\frac{q^{2(t_1-D)}}{N_{q^2}(m, t_1 - D)} \cdot \frac{q^{2t_2}}{N_{q^2}(m, t_2)} \cdot \min(S_{q^2}(m, t_1 - D), S_{q^2}(m, t_2)). \tag{3.7}$$

The expected number of distinct irreducible factors of R_1/Q and R_2 is $A_{q^2}(m, t_1 - D) + A_{q^2}(m, t_2)$. In the analysis, we shall assume that each of these irreducible factors has degree exactly m.

3.3.6 QPA descent

The QPA descent method is so named because it was a crucial step in the Barbulescu et al. quasi-polynomial time algorithm for the DLP in finite fields of small characteristic [16].

Let $Q \in \mathbb{F}_{q^2}[X]$ with $\deg Q = D$, and let $m \in [\lceil D/2 \rceil, D - 1]$. Let $(a, b, c, d) \in \mathcal{P}_q$, and recall that $|\mathcal{P}_q| = q^3 + q$. Substituting $Y \mapsto (aQ + b)/(cQ + d)$ into the systematic equation (3.2) and multiplying by $(cQ + d)^{q+1}$ yields

$$(aQ + b)^q (cQ + d) - (aQ + b)(cQ + d)^q = (cQ + d) \prod_{\alpha \in \mathbb{F}_q} [(a - \alpha c)Q + (b - \alpha d)]. \tag{3.8}$$

The left side of (3.8) can be written as

$$\begin{aligned}(\overline{a}\overline{Q}(X^q) + \overline{b})(cQ + d) - (aQ + b)(\overline{c}\overline{Q}(X^q) + \overline{d}) \\ \equiv (\overline{a}\overline{Q}(\frac{h_0}{h_1}) + \overline{b})(cQ + d) - (aQ + b)(\overline{c}\overline{Q}(\frac{h_0}{h_1}) + \overline{d}) \pmod{I_X}.\end{aligned}$$

Hence

$$(\overline{a}\overline{Q}(\frac{h_0}{h_1}) + \overline{b})(cQ+d) - (aQ+b)(\overline{c}\overline{Q}(\frac{h_0}{h_1}) + \overline{d}) \tag{3.9}$$
$$\equiv (cQ+d) \prod_{\alpha \in \mathbb{F}_q} [(a-\alpha c)Q + (b - \alpha d)] \pmod{I_X}.$$

Multiplying (3.9) by h_1^D yields

$$(\overline{a}\widetilde{Q} + \overline{b}h_1^D)(cQ+d) - (aQ+b)(\overline{c}\widetilde{Q} + \overline{d}h_1^D) \tag{3.10}$$
$$\equiv h_1^D \cdot (cQ+d) \cdot \prod_{\alpha \in \mathbb{F}_q} [(a-\alpha c)Q + (b - \alpha d)] \pmod{I_X},$$

where $\widetilde{Q}(X) = h_1^D \cdot \overline{Q}(h_0/h_1)$. Note that the polynomial on the left side of (3.10) has degree at most $3D$. If this polynomial is m-smooth, then (3.10) yields a linear relation of the logarithms of some degree-m polynomials and logarithms of translates of Q. After collecting slightly more than q^2 such relations, one searches for a linear combination of these relations that eliminates all translates of Q except for Q itself. To achieve this, consider row vectors in $(\mathbb{Z}_N)^{q^2}$ with coordinates indexed by elements $\lambda \in \mathbb{F}_{q^2}$. For each relation, we define a vector v whose entry v_λ is 1 if $Q - \lambda$ appears in the right side of (3.10), and 0 otherwise. If the resulting matrix $\mathcal{H}(Q)$ of row vectors has full column rank, then one obtains an expression for $\log_g Q$ in terms of the logarithms of polynomials of degree at most m. The number of distinct polynomials of degree at most m in this expression is expected to be $A_{q^2}(m, 3D) \cdot q^2$; in the analysis we shall assume that each of these polynomials has degree exactly m.

Since the probability that a degree-$3D$ polynomial is m-smooth is $N_{q^2}(m, 3D)/\ (q^2)^{3D}$, one must have

$$\frac{N_{q^2}(m, 3D)}{q^{6D}} \cdot (q^3 + q) \gg q^2 \tag{3.11}$$

in order to ensure that $\mathcal{H}(Q)$ has much more than q^2 rows, whereby $\mathcal{H}(Q)$ can be expected to have full rank.

The expected cost of the relation generation portion of QPA descent is

$$\frac{q^{6D}}{N_{q^2}(m, 3D)}\, q^2 \cdot S_{q^2}(m, 3D),$$

while the cost of the linear algebra is $q^5 \cdot A_N$.

3.3.7 Gröbner bases descent

Let $Q \in \mathbb{F}_{q^2}[X]$ with $\deg Q = D$, and let $m = \lceil (D+1)/2 \rceil$. In Joux's new descent method [77, §5.3], one finds degree-m polynomials[2] $k_1, k_2 \in \mathbb{F}_{q^2}[X]$ such that $Q \mid G$, where

$$G = h_1^m (k_1^q k_2 - k_1 k_2^q) \bmod I_X.$$

[2]More generally, the degrees of k_1 and k_2 can be different.

We then have

$$h_1^m \cdot k_2 \cdot \prod_{\alpha \in \mathbb{F}_q} (k_1 - \alpha k_2) \equiv G(X) \pmod{I_X}$$

as can be seen by making the substitution $Y \mapsto k_1/k_2$ into the systematic equation (3.2) and clearing denominators. Define $\widetilde{k}(X) = h_1^m \cdot \overline{k}(h_0/h_1)$ and note that $\deg \widetilde{k} = 2m$. We thus have $G \equiv \widetilde{k}_1 k_2 - k_1 \widetilde{k}_2 \pmod{I_X}$, and consequently $G = \widetilde{k}_1 k_2 - k_1 \widetilde{k}_2$ provided that $3m < n$. It follows that $G(X) = Q(X)R(X)$ for some $R \in \mathbb{F}_{q^2}[X]$ with $\deg R = 3m - D$. If R is m-smooth, we obtain a linear relationship between $\log_g Q$ and logarithms of degree-m polynomials by taking logarithms of both sides of the following:

$$h_1^m \cdot k_2 \cdot \prod_{\alpha \in \mathbb{F}_q} (k_1 - \alpha k_2) \equiv Q(X)R(X) \pmod{I_X}. \tag{3.12}$$

To determine (k_1, k_2, R) that satisfy

$$\widetilde{k}_1 k_2 - k_1 \widetilde{k}_2 = Q(X)R(X), \tag{3.13}$$

one can transform (3.13) into a system of multivariate bilinear equations over $\mathbb{F}_q$. Specifically, each coefficient of k_1, k_2 and R is written using two variables over $\mathbb{F}_q$, the two variables representing the real and imaginary parts of that coefficient (which is in $\mathbb{F}_{q^2}$). The coefficients of $\widetilde{k}_1$ and $\widetilde{k}_2$ can then be written in terms of the coefficients of k_1 and k_2. Hence, equating coefficients of X^i of both sides of (3.13) yields $3m + 1$ quadratic equations. The real and imaginary parts of each of these equations are equated, yielding $6m + 2$ bilinear equations in $10m - 2D + 6$ variables over $\mathbb{F}_q$. This system of equations can be solved by finding a Gröbner basis for the ideal it generates. Finally, solutions (k_1, k_2, R) are tested until one is found for which R is m-smooth. This yields an expression for $\log_g Q$ in terms of the logarithms of approximately $q + 1 + A_{q^2}(m, 3m - D)$ polynomials of degree (at most) m; in the analysis we shall assume that each of the polynomials has degree exactly m.

Now, the number of candidate pairs (k_1, k_2) is $((q^2)^{m+1})^2 = q^{4(m+1)}$. Since $(q^2)^{3m-D+1}$ of the $(q^2)^{3m+1}$ degree-$(3m)$ polynomials in $\mathbb{F}_{q^2}[X]$ are divisible by $Q(X)$, the number of solutions (k_1, k_2, R) is expected to be approximately

$$\frac{q^{2(3m-D+1)}}{q^{2(3m+1)}} \cdot q^{4(m+1)} = q^{4(m+1)-2D}.$$

However, the number of distinct R obtained will be much less than $q^{4(m+1)-2D}$. For example, any two pairs (k_1', k_2') and (k_1'', k_2'') with $k_1'/k_2' = k_1''/k_2''$ will generate the same R, so the expected number of distinct R is at most $q^{4(m+1)-2D}/(q^2 - 1)$. Let us denote by $R(m, D)$ the expected number of distinct R obtainable. Then the condition

$$R(m, D) \gg \frac{q^{2(3m-D)}}{N_{q^2}(m, 3m - D)}, \tag{3.14}$$

can ensure that there exists a solution (k_1, k_2, R) for which R is m-smooth.

The number $R(m, D)$ has not been determined in general. For the case $m = 1$ and $D = 2$, one must select $k_1 = aX + b$ and $k_2 = cX + d$ with $(a, b, c, d) \in \mathcal{P}_q$ to avoid collisions; hence $R(1, 2) \leq \frac{q^4}{q^8}(q^3 + q) \approx \frac{1}{q}$ and descending from 2 to 1 can be expected to succeed only for 1 out of every q quadratics; this is indeed what we observed in our experiments. In general, the success of the Gröbner bases descent step is best determined experimentally (cf. §3.4.7).

It is difficult to determine the exact cost $G_{q^2}(m, D)$ of the Gröbner basis finding step. After the Gröbner basis is found, the cost to find an m-smooth R is $(q^2)^{3m-D}/N_{q^2}(m, 3m - D) \cdot S_{q^2}(m, 3m - D)$.

3.4 Computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 509}}$

We present a concrete analysis of the DLP algorithm described in §3.3 for computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 509}}$. In fact, this field is embedded in the quadratic extension field $\mathbb{F}_{3^{12\cdot 509}}$, and it is the latter field where the DLP algorithm of §3.3 is executed. Thus, we have $q = 3^6 = 729$, $n = 509$, and $N = 3^{12\cdot 509} - 1$. Note that $3^{12\cdot 509} \approx 2^{9681}$. We wish to find $\log_g h$, where g is a generator of $\mathbb{F}^*_{3^{12\cdot 509}}$ and $h \in \mathbb{F}^*_{3^{12\cdot 509}}$.

As mentioned in §3.1, our main motivation for finding discrete logarithms in $\mathbb{F}_{3^{6\cdot 509}}$ is to attack the elliptic curve discrete logarithm problem in $E(\mathbb{F}_{3^{509}})$, where E is the supersingular elliptic curve $y^2 = x^3 - x + 1$ with $|E(\mathbb{F}_{3^{509}})| = 7r$, and where $r = (3^{509} - 3^{255} + 1)/7$ is an 804-bit prime. Note that $r^2 \nmid N$. The elliptic curve discrete logarithm problem in the order-r subgroup of $E(\mathbb{F}_{3^{509}})$ can be efficiently reduced to the discrete logarithm problem in the order-r subgroup of $\mathbb{F}^*_{3^{12\cdot 509}}$. In the latter problem, we are given two elements α, β of order r in $\mathbb{F}^*_{3^{12\cdot 509}}$ and we wish to find $\log_\alpha \beta$. It can readily be seen that $\log_\alpha \beta = (\log_g \beta)/(\log_g \alpha) \bmod r$. Thus, we will henceforth assume that h has order r and that we only need to find $\log_g h \bmod r$. An immediate consequence of this restriction is that all the linear algebra in the new algorithm has to be performed modulo the 804-bit r instead of modulo the 9681-bit N.

The parameters for each step of the algorithm were carefully chosen in order to balance the running time of the steps. We also took into account the degree to which each step could be parallelized on conventional computers. A summary of the parameter choices for the descent is given in Figure 3.1. The costs of each step are given in Table 3.3.

Recall from §2.5.1 that the total cost for testing m-smoothness of a degree-d polynomial in $\mathbb{F}_q[X]$ is

$$S_{3^{12}}(m, d) = 2d^2(d + m + 7) \quad \mathbb{F}_{3^{12}}\text{-multiplications.} \tag{3.15}$$

3.4.1 Setup

We chose the representations

$$\mathbb{F}_{3^6} = \mathbb{F}_3[U]/(U^6 + 2U^4 + U^2 + 2U + 2)$$

and

$$\mathbb{F}_{3^{12}} = \mathbb{F}_{3^6}[V]/(V^2 + U^{365}).$$

We selected

$$h_0 = (U^{553}V + U^{343})X^2 + (U^{535}V + U^{417})X + (U^{172}V + U^{89}) \in \mathbb{F}_{3^{12}}[X]$$

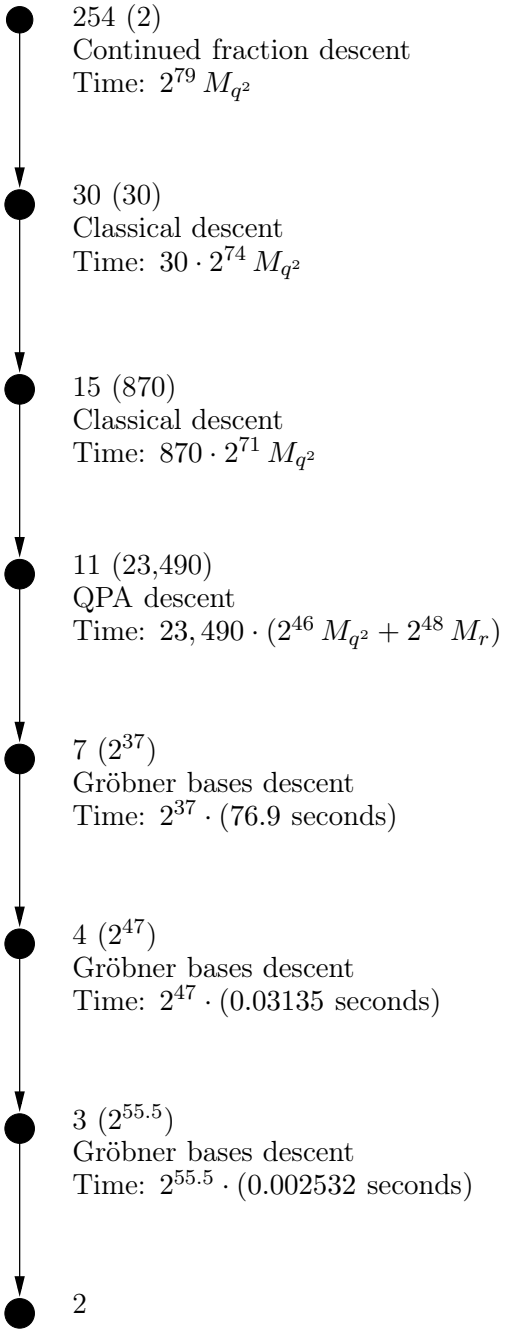

Figure 3.1: A typical path of the descent tree for computing an individual logarithm in $\mathbb{F}_{3^{12\cdot 509}}$ ($q = 3^6$). The numbers in parentheses next to each node are the expected number of nodes at that level. 'Time' is the expected time to generate all nodes at a level.

Finding logarithms of linear polynomials		
Relation generation	$2^{30}M_{q^2}$	$2^{30}M_{q^2}$
Linear algebra	$2^{48}A_r$	$2^{50}M_{q^2}$
Finding logarithms of irreducible quadratic polynomials		
Relation generation	$3^{12}\cdot 2^{39}M_{q^2}$	$2^{58}M_{q^2}$
Linear algebra	$3^{12}\cdot 2^{48}A_r$	$2^{69}M_{q^2}$
Descent		
Continued-fraction (254 to 30)	$2^{79}M_{q^2}$	$2^{79}M_{q^2}$
Classical (30 to 15)	$30\cdot 2^{74}M_{q^2}$	$2^{79}M_{q^2}$
Classical (15 to 11)	$870\cdot 2^{71}M_{q^2}$	$2^{81}M_{q^2}$
QPA (11 to 7)	$23,490\cdot(2^{46}M_{q^2}+2^{48}A_r)$	$2^{65}M_{q^2}$
Gröbner bases (7 to 4)	$2^{37}\cdot$ (76.9 seconds)	$2^{73}M_{q^2}$
Gröbner bases (4 to 3)	$2^{47}\cdot$ (0.03135 seconds)	$2^{72}M_{q^2}$
Gröbner bases (3 to 2)	$2^{55.5}\cdot$ (0.002532 seconds)	$2^{77}M_{q^2}$

Table 3.3: Estimated costs of the main steps of the new DLP algorithm for computing discrete logarithms in $\mathbb{F}_{3^{12\cdot 509}}$ ($q=3^6$). A_r and M_{q^2} denote the costs of an addition modulo the 804-bit prime $r=(3^{509}-3^{255}+1)/7$ and a multiplication in $\mathbb{F}_{3^{12}}$. We use the cost ratio $A_r/M_{q^2}=4$, and also assume that 2^{30} multiplications in $\mathbb{F}_{3^{12}}$ can be performed in 1 second (cf. §3.4.8).

and $h_1=1$, and $I_X\in\mathbb{F}_{3^{12}}[X]$ to be the degree-509 monic irreducible factor of $X^{3^6}-h_0$. The other irreducible factors have degrees 43, 55 and 122.

3.4.2 Finding logarithms of linear polynomials

The factor base $\mathcal{B}_1$ has size $3^{12}\approx 2^{19}$. The cost of relation generation is approximately $2^{30}M_{q^2}$, whereas the cost of the linear algebra is approximately $2^{48}A_r$.

3.4.3 Finding logarithms of irreducible quadratic polynomials

For each $u\in\mathbb{F}_{3^{12}}$, the expected cost of computing logarithms of all quadratics in $\mathcal{B}_{2,u}$ is $2^{39}M_{q^2}$ for the computation of $\mathcal{H}(Q)$, and $2^{48}A_r$ for the linear algebra. Note that the number of columns in $\mathcal{H}(Q)$ can be halved since the logarithms of all reducible quadratics are known. Since the expected number of relations obtainable is

$$\frac{N_{q^2}(1,6)}{q^{12}}\cdot(q^3+q)\approx\frac{1}{719.98}\cdot(q^3+q)\approx q^2+6659,$$

one can expect that the matrix $\mathcal{H}(Q)$ will have full rank.

3.4.4 Continued-fraction descent

For the continued-fraction descent, we selected $m=30$. The expected cost of this descent is $2^{79}M_{q^2}$. The expected number of distinct irreducible factors of degree (at most) 30 obtained is $2A_{3^{12}}(30,254)\approx 30$.

3.4.5 Classical descent

Two classical descent stages are employed. In the first stage, we have $D = 30$ and select $m = 15$, $s = 3$, $d_1 = 5$, $d_2 = 5$, and $\delta = 4$. The set of $D + \delta$ pairs (i, j) selected was

$$([0, 3] \times [0, 5]) \cup \{(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 0), (5, 1), (5, 2), (5, 3), (5, 4)\},$$

yielding $t_1 = 138$ and $t_2 = 143$. Note that inequality (3.6) is satisfied. The expected cost of the descent for each of the 30 degree-30 polynomials is approximately $2^{52} \cdot S_{q^2}(15, 108)$. The expected total number of distinct irreducible polynomials of degree (at most) 15 obtained is approximately 870.

In the second classical descent stage, we have $D = 15$ and select $m = 11$, $s = 3$, $d_1 = 3$, $d_2 = 4$, and $\delta = 4$. The set of $D + \delta$ pairs (i, j) selected was

$$([0, 2] \times [0, 4]) \cup \{(3, 0), (3, 1), (3, 2), (3, 3)\},$$

yielding $t_1 = 110$ and $t_2 = 87$. Note that inequality (3.6) is satisfied. The expected cost of the descent for each of the 870 degree-15 polynomials is approximately $2^{50} \cdot S_{q^2}(11, 87)$. The expected total number of distinct irreducible polynomials of degree (at most) 11 obtained is approximately 23,490.

3.4.6 QPA descent

The QPA descent method is then applied to each of the 23,490 degree-11 polynomials Q obtained from the classical descent stage. We have $D = 11$ and $m = 7$. For each Q, the expected number of rows in $\mathcal{H}(Q)$ is 570,172, so we can expect this matrix to have full column rank (namely, $q^2 = 531,441$). For each Q, the expected cost of relation generation is $2^{29} \cdot S_{q^2}(7, 33)$ and the cost of the linear algebra is $2^{48} A_r$. Also for each Q, the expected number of distinct polynomials of degree at most 7 obtained is expected to be $A_{q^2}(7, 33) \cdot q^2 \approx 2^{22}$. Thus, the total number of distinct polynomials of degree at most 7 obtained after the QPA descent stage is approximately 2^{37}.

3.4.7 Gröbner bases descent

The Gröbner bases descent method is applied to each of the 2^{37} polynomials of degree (at most) 7 obtained after QPA descent. Our experiments were run using Magma v2.19-7 [98] on a 2.9 GHz Intel core i7-3520M.

First, one descends from 7 to 4, i.e., $D = 7$ and $m = 4$. For each degree-7 polynomial Q, we have to solve a system of 26 quadratic polynomial equations in 32 variables over $\mathbb{F}_q$ (cf. (3.13)). Since the ideal generated by these polynomials typically has dimension greater than 0, we randomly fix some of the variables in the hope of obtaining a 0-dimensional ideal. (More precisely, we added some linear constraints involving pairs of variables, one variable from k_1 and the other from k_2.) Each degree-5 R obtained from the variety of the resulting ideal is tested for 4-smoothness. If no 4-smooth R is obtained, we randomly fix some other subset of variables and repeat. We ran 17,510 Gröbner bases descent experiments with randomly-selected degree-7 polynomials Q. On average, we had to find 1.831 Gröbner bases for each Q. The average number of R's tested for 4-smoothness for each Q was 1.252, which agrees with

the expected number $q^{10}/N_{q^2}(4,5) \approx 1.25$. The average time to find each Gröbner basis was 42.0 seconds, and the memory consumption was 64 Mbytes. In total, the expected number of polynomials of degree at most 4 obtained is $2^{37}(q+1+A_{q^2}(4,5)) \approx 2^{47}$.

Next, one descends from 4 to 3, i.e., $D = 4$ and $m = 3$. For each degree-4 polynomial Q, we have to solve a system of 20 quadratic polynomial equations in 28 variables over $\mathbb{F}_q$. We proceed as above, by fixing some of the 28 variables. We ran 1,230,000 Gröbner bases descent experiments with randomly-selected degree-4 polynomials Q. On average, we had to find 2.361 Gröbner bases for each Q. The average number of R's tested for 3-smoothness for each Q was 1.815, which agrees with the expected number $q^{10}/N_{q^2}(3,5) \approx 1.818$. The average time to find each Gröbner basis was 0.01328 seconds, and the memory consumption was 32 Mbytes. In total, the expected number of polynomials of degree at most 3 obtained is $2^{47}(q+1+A_{q^2}(3,5)) \approx 2^{57}$.

Finally, one descends from 3 to 2, i.e., $D = 3$ and $m = 2$. Since the total number of monic irreducible cubics over $\mathbb{F}_{q^2}$ is approximately $2^{55.5}$, which is less than 2^{57}, we perform the 3 to 2 descent for all monic irreducible cubics. For each such polynomial Q, we have to solve a system of 14 quadratic polynomial equations in 20 variables over $\mathbb{F}_q$. We proceed as above, by fixing some of the 20 variables. We ran 8,100,000 Gröbner bases descent experiments with randomly-selected degree-3 polynomials Q. On average, we had to find 2.026 Gröbner bases for each Q. The average number of R's tested for 2-smoothness for each Q was 1.499, which agrees with the expected number $q^6/N_{q^2}(2,3) \approx 1.5$. The average time to find each Gröbner basis was 0.00125 seconds, and the memory consumption was 32 Mbytes.

3.4.8 Overall running time

The second column of Table 3.3 gives the running time estimates for the main steps of the new DLP algorithm in three units of time: A_r, M_{q^2}, and seconds. In order to assess the overall time, we make some assumptions about the ratios of these units of time.

First, we shall assume that $A_r/M_{q^2} = 4$. To justify this, we observe that an 804-bit integer can be stored in thirteen 64-bit words. The X86-64 instruction set has an **ADD** operation that adds two 64-bit unsigned integers in one clock cycle. Hence, integer addition can be completed in 13 clock cycles. Modular reductions comprises one conditional statement plus one subtraction (required in roughly half of all modular additions). One can use a lazy reduction technique that amortizes the cost of a modular reduction among many integer additions. All in all, the cost of A_r can be estimated to be 13 clock cycles. Unlike for 64-bit integer multiplication, there is no native support for $\mathbb{F}_{3^{12}}$ multiplication on an Intel Core i7 machine. However, we expect that a specially designed multiplier could be built to achieve a multiplication cost of 4 clock cycles. While building such a native multiplier would certainly be costly, this expense can be expected to be within the budget of a well-funded adversary who is contemplating implementing the new DLP algorithm. This gives us an A_r/M_{q^2} ratio of approximately 4.

Next, since a multiplication in $\mathbb{F}_{3^{12}}$ can be done in 4 clock cycles, we will transform one second on a 2.9 GHz machine (on which the Gröbner bases descent experiments were performed) into $2^{30}M_{q^2}$.

Using these estimates, we see from the third column of Table 3.3 that the overall running

time of the new algorithm is approximately $2^{81.7}M_{q^2}$. We note that the relation generation, continued-fraction descent, classical descent, and Gröbner bases descent steps, and also the relation generation portion of QPA descent, are effectively parallelizable in the sense that one can essentially achieve a factor-C speedup if C processors are available. Using the experimental results in [74, 14] as a guide, we can safely estimate that each linear system of equations can be solved in less than one day of using a small number of GPUs and CPUs. Thus, we conclude that the linear system of equations for finding logarithms of linear polynomials, the $3^{12} \approx 2^{19}$ linear systems of equations for finding logarithms of irreducible quadratic polynomials, and the $23,490$ linear systems of equations in QPA can be effectively parallelized on conventional computers.

Remark 3.2. (*caveat emptor*) Although our analysis is concrete rather than asymptotic, it must be emphasized that the analysis makes several heuristic assumptions and approximations. For example, there are the usual heuristic assumptions that certain polynomials encountered are uniformly distributed over the set of all polynomials of the same degree. Furthermore, we have assumed that the matrix $\mathcal{H}(Q)$ in QPA descent indeed has full column rank. Also, our run time analysis ignores operations such as additions in $\mathbb{F}_{3^{12}}$ and memory accesses. Thus, further analysis and experimentation is needed before one can conclude with certainty that the $2^{81.7}M_{q^2}$ running time estimate is an accurate measure of the efficiency of the new DLP algorithm for computing logarithms in the order-r subgroup of $\mathbb{F}^*_{3^{6\cdot 509}}$.

Remark 3.3. (*looseness of our upper bound on the running time*) Remark 3.2 notwithstanding, our analysis is quite conservative and there are several possible ways in which the upper bound on the running time could be improved. (i) In our estimates for the number of branches in a descent step, we assume that each distinct irreducible polynomial obtained has degree exactly m, whereas in practice many of these polynomials will have degree significantly less than m. Thus, it would appear that our upper bound on the number of nodes in the descent tree is quite loose. (ii) The Gröbner bases descent running times reported in §3.4.7 can be expected to be significantly improved by a native implementation of the F4 [47] or F5 [48] Gröbner basis finding algorithms optimized for characteristic-three finite fields. (Magma implements the F4 algorithm, but is not optimized for characteristic-three finite fields.) (iii) An optimized Gröbner basis implementation might be successful in performing the descent from $D = 11$ to $D = 6$, thereby replacing the QPA descent from $D = 11$ to $D = 7$ and significantly reducing the number of nodes in the descent tree. (iv) Bernstein's smoothness testing method [24] might be faster in practice than the basic method described in Section 2.5. (v) Sieving can be expected to significantly speedup the continued-fraction descent stage [15].

3.4.9 Comparisons with Joux-Lercier

Shinohara et al. [115] estimated that the running time of the Joux-Lercier algorithm [81] for computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 509}}$ is $2^{111.35}$ for the relation generation stage, and $2^{102.69}$ for the linear algebra stage; the units of time were not specified. The relation generation time can be significantly decreased using Joux's pinpointing technique [76] without having

a noticeable impact on the linear algebra time. We note also that the linear algebra cost of $2^{102.69}$ is an underestimation since it does not account for the number of nonzero coefficients in each equation. In any case, since the relation generation is effectively parallelizable on conventional computers whereas the linear algebra is not, the linear algebra stage is the dominant step of the Joux-Lercier algorithm. Due to its large size, the linear algebra stage will remain infeasible for the foreseeable future.

In contrast, the new algorithm is effectively parallelizable and has an overall running time of $2^{81.7}M_{q^2}$. If one had access to a massive number of processors (e.g., 2^{30} processors), then the new algorithm could be executed within one year.

We believe that these comparisons justify the claim made in Section 3.1 about the weakness of the field $\mathbb{F}_{3^{6\cdot 509}}$, and thereby also the supersingular elliptic curve over $\mathbb{F}_{3^{509}}$ with embedding degree 6.

3.5 Computing discrete logarithms in $\mathbb{F}_{2^{12\cdot 367}}$

We present a concrete analysis of the DLP algorithm described in §3.3 for computing discrete logarithms in $\mathbb{F}_{2^{12\cdot 367}}$. In fact, this field is embedded in the quadratic extension field $\mathbb{F}_{2^{24\cdot 367}}$, and it is the latter field where the DLP algorithm of §3.3 is executed. Thus, we have $q = 2^{12}$, $n = 367$, and $N = 2^{24\cdot 367} - 1$. Note that $2^{24\cdot 367} \approx 2^{8808}$. We wish to find $\log_g h$, where g is a generator of $\mathbb{F}^*_{2^{24\cdot 367}}$ and $h \in \mathbb{F}^*_{2^{24\cdot 367}}$.

As mentioned in §3.1, our main motivation for finding discrete logarithms in $\mathbb{F}_{2^{12\cdot 367}}$ is to attack the discrete logarithm problem in $\mathrm{Jac}_C(\mathbb{F}_{2^{367}})$, where C is the supersingular genus-2 curve $y^2 + y = x^5 + x^3$ with $|\mathrm{Jac}_C(\mathbb{F}_{2^{367}})| = 13 \cdot 7170258097 \cdot r$, and where $r = (2^{734} + 2^{551} + 2^{367} + 2^{184} + 1)/(13 \cdot 7170258097)$ is a 698-bit prime. Note that $r^2 \nmid N$. The discrete logarithm problem in the order-r subgroup of $\mathrm{Jac}_C(\mathbb{F}_{2^{367}})$ can be efficiently reduced to the discrete logarithm problem in the order-r subgroup of $\mathbb{F}^*_{2^{12\cdot 367}}$. Thus, we will henceforth assume that h has order r and that we only need to find $\log_g h \bmod r$. An immediate consequence of this restriction is that all the linear algebra in the new algorithm has to be performed modulo the 698-bit r instead of modulo the 8808-bit N.

The parameters for each step of the algorithm were chosen in order to balance the running time of the steps. We also took into account the degree to which each step could be parallelized on conventional computers. A summary of the parameter choices for the descent is given in Figure 3.2. The costs of each step are given in Table 3.4.

If $f \in \mathbb{F}_q[X]$ has degree d, then $X^q \bmod f$ can be determined by first precomputing $X^4, X^8, \ldots, X^{4(d-1)} \bmod f$ by repeated multiplication by X. Thereafter, computing the fourth power of a polynomial modulo f can be accomplished by computing fourth powers of the coefficients of the polynomial, and then multiplying the precomputed polynomials by these fourth powers (and adding the results). In this way, we get a loose upper bound of $4d^2 + 11d^2 = 15d^2$ $\mathbb{F}_{2^{24}}$-multiplications of the cost to compute $X^{2^{24}} \bmod f$, and the total cost for testing m-smoothness of f, with respect to the method in §2.5.1, becomes

$$S_{2^{24}}(m, d) = 2d^2(d + m + 7.5) \quad \mathbb{F}_{2^{24}}\text{-multiplications.} \tag{3.16}$$

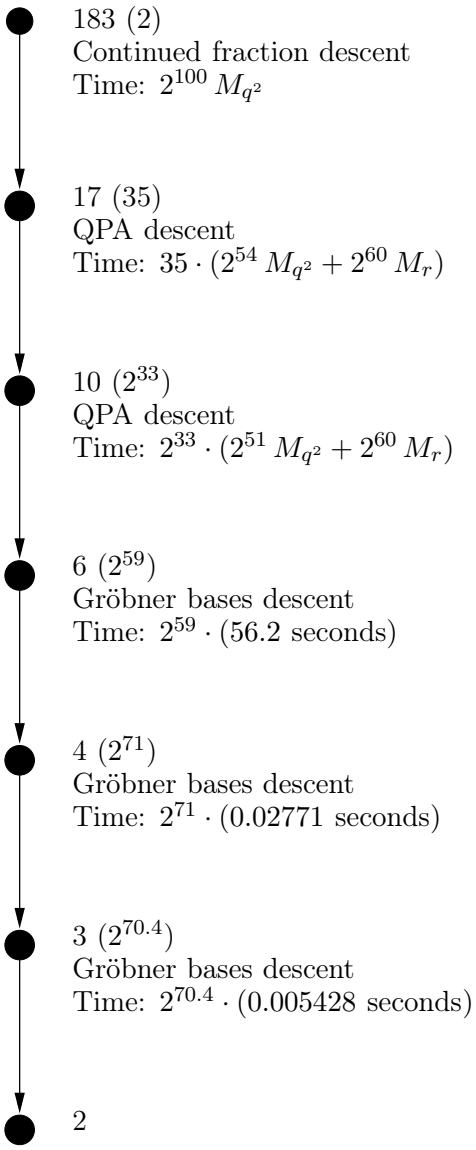

Figure 3.2: A typical path of the descent tree for computing an individual logarithm in $\mathbb{F}_{2^{24\cdot367}}$ ($q = 2^{12}$). The numbers in parentheses next to each node are the expected number of nodes at that level. 'Time' is the expected time to generate all nodes at a level.

Finding logarithms of linear polynomials		
Relation generation	$2^{35}M_{q^2}$	$2^{35}M_{q^2}$
Linear algebra	$2^{60}A_r$	$2^{60}M_{q^2}$
Finding logarithms of irreducible quadratic polynomials		
Relation generation	$2^{24}\cdot 2^{44}M_{q^2}$	$2^{68}M_{q^2}$
Linear algebra	$2^{24}\cdot 2^{60}A_r$	$2^{84}M_{q^2}$
Descent		
Continued-fraction (183 to 17)	$2^{100}M_{q^2}$	$2^{100}M_{q^2}$
QPA (17 to 10)	$35\cdot(2^{54}M_{q^2}+2^{60}A_r)$	$2^{65}M_{q^2}$
QPA (10 to 6)	$2^{33}\cdot(2^{51}M_{q^2}+2^{60}A_r)$	$2^{93}M_{q^2}$
Gröbner bases (6 to 4)	$2^{59}\cdot$ (64.9 seconds)	$2^{93}M_{q^2}$
Gröbner bases (4 to 3)	$2^{71}\cdot$ (0.02771 seconds)	$2^{94}M_{q^2}$
Gröbner bases (3 to 2)	$2^{70.4}\cdot$ (0.005428 seconds)	$2^{91}M_{q^2}$

Table 3.4: Estimated costs of the main steps of the new DLP algorithm for computing discrete logarithms in $\mathbb{F}_{2^{24\cdot 367}}$ ($q=2^{12}$). A_r and M_{q^2} denote the costs of an addition modulo the 698-bit prime $r=(2^{734}+2^{551}+2^{367}+2^{184}+1)/(13\cdot 7170258097)$ and a multiplication in $\mathbb{F}_{2^{24}}$. We use the cost ratio $A_r/M_{q^2}=1$, and also assume that 2^{28} multiplications in $\mathbb{F}_{2^{24}}$ can be performed in 1 second (cf. §3.5.8).

3.5.1 Setup

We chose the representations

$$\mathbb{F}_{2^{12}}=\mathbb{F}_2[U]/(U^{12}+U^7+U^6+U^5+U^3+U+1)$$

and

$$\mathbb{F}_{2^{24}}=\mathbb{F}_{2^{12}}[V]/(V^2+U^{152}V+U^{3307}).$$

We selected $h_0=(U^{2111}V+U^{2844})X^2+(U^{428}V+U^{2059})X+(U^{1973}V+U^{827})\in\mathbb{F}_{2^{24}}[X]$ and $h_1=X+U^{2904}V+U^{401}\in\mathbb{F}_{2^{24}}[X]$, and $I_X\in\mathbb{F}_{2^{24}}[X]$ to be the degree-367 monic irreducible factor of $h_1X^{2^{12}}-h_0$. The other irreducible factors of $h_1X^{2^{12}}-h_0$ have degrees 23, 103, 162, 298 and 3144.

3.5.2 Finding logarithms of linear polynomials

The factor base $\mathcal{B}_1$ has size 2^{24}. The cost of relation generation is approximately $2^{35}M_{q^2}$, whereas the cost of the linear algebra is approximately $2^{60}A_r$.

3.5.3 Finding logarithms of irreducible quadratic polynomials

For each $u\in\mathbb{F}_{2^{24}}$, the expected cost of computing logarithms of all quadratics in $\mathcal{B}_{2,u}$ is $2^{44}M_{q^2}$ for the computation of $\mathcal{H}(Q)$, and $2^{60}A_r$ for the linear algebra.

3.5.4 Continued-fraction descent

We selected $m=17$. The expected cost of this descent is $2^{100}M_{q^2}$. The expected number of distinct irreducible factors of degree (at most) 17 obtained is $2A_{2^{24}}(17,183)\approx 35$.

3.5.5 Classical descent

When applicable, classical descent is preferable to QPA descent since the former produces a far smaller number of branches when descending from a polynomial Q. However, in the field under consideration we have $q = 2^{12}$, so at least one of X^{2^s} and $X^{2^{12-s}}$ has degree at least 64. This means that at least one of the polynomials $R_1 = R(X, X^{2^s})$ and $R_2 = R'(X^{2^{12-s}}, h_0)$ (cf. §3.3.5) has very large degree, rendering classical descent ineffective.

3.5.6 QPA descent

The QPA descent method is applied to each of the 35 degree-17 polynomials Q obtained from the continued-fraction descent stage. We have $D = 17$ and $m = 10$. For each Q, the expected cost of relation generation is $2^{54}M_{q^2}$ and the cost of the linear algebra is $2^{60}A_r$. Also for each Q, the expected number of distinct polynomials of degree at most 6 obtained is expected to be $A_{q^2}(10, 51) \cdot q^2 \approx 2^{28}$. Thus, the total number of distinct polynomials of degree at most 10 obtained after the first QPA descent stage is approximately 2^{33}.

The QPA descent method is then applied to each of these 2^{33} degree-10 polynomials Q. We have $D = 10$ and $m = 6$. For each Q, the expected cost of relation generation is $2^{51}M_{q^2}$ and the cost of the linear algebra is $2^{60}A_r$. Also for each Q, the expected number of distinct polynomials of degree at most 6 obtained is expected to be $A_{q^2}(6, 30) \cdot q^2 \approx 2^{26}$. Thus, the total number of distinct polynomials of degree at most 6 obtained after the second QPA descent stage is approximately 2^{59}.

3.5.7 Gröbner bases descent

The Gröbner bases descent method is applied to each of the 2^{59} polynomials of degree (at most) 6 obtained after QPA descent. Our experiments were run using Magma v2.19-7 [98] on a 2.9 GHz Intel core i7-3520M.

First, one descends from 6 to 4, i.e., $D = 6$ and $m = 4$. For each degree-6 polynomial Q, we have to solve a system of 26 quadratic polynomial equations in 34 variables over $\mathbb{F}_q$ (cf. (3.13)). After fixing some variables, each degree-6 R obtained from the variety of the resulting ideal is tested for 4-smoothness. If no 4-smooth R is obtained, we randomly fix some other subset of variables and repeat. We ran 11,810 Gröbner bases descent experiments with randomly-selected degree-6 polynomials Q. On average, we had to find 2.112 Gröbner bases for each Q. The average number of R's tested for 4-smoothness for each Q was 1.585, which agrees with the expected number $q^{12}/N_{q^2}(4, 6) \approx 1.579$. The average time to find each Gröbner basis was 30.74 seconds. In total, the expected number of polynomials of degree at most 4 obtained is $2^{59}(q + 1 + A_{q^2}(4, 6)) \approx 2^{71}$.

Next, one descends from 4 to 3, i.e., $D = 4$ and $m = 3$. For each degree-4 polynomial Q, we have to solve a system of 20 quadratic polynomial equations in 28 variables over $\mathbb{F}_q$. We proceed as above, by fixing some of the 28 variables. We ran 3,608,000 Gröbner bases descent experiments with randomly-selected degree-4 polynomials Q. On average, we had to find 2.362 Gröbner bases for each Q. The average number of R's tested for 3-smoothness for each Q was 1.817, which agrees with the expected number $q^{10}/N_{q^2}(3, 5) \approx 1.818$. The average time to find each Gröbner basis was 0.01173 seconds. In total, the expected number

of polynomials of degree at most 3 obtained is $2^{71}(q+1+A_{q^2}(3,5)) \approx 2^{83}$.

Finally, one descends from 3 to 2, i.e., $D=3$ and $m=2$. Since the total number of monic irreducible cubics over $\mathbb{F}_{q^2}$ is approximately $2^{70.4}$, which is less than 2^{83}, we perform the 3 to 2 descent for all monic irreducible cubics. For each such polynomial Q, we have to solve a system of 14 quadratic polynomial equations in 20 variables over $\mathbb{F}_q$. We proceed as above, by fixing some of the 20 variables. We ran 1,080,000 Gröbner bases descent experiments with randomly-selected degree-3 polynomials Q. On average, we had to find 2.024 Gröbner bases for each Q. The average number of R's tested for 2-smoothness for each Q was 1.5, which agrees with the expected number $q^6/N_{q^2}(2,3) \approx 1.5$. The average time to find each Gröbner basis was 0.002682 seconds.

3.5.8 Overall running time

In order to assess the overall time, we make some assumptions about the ratios of units of time used in Table 3.4, namely A_r, M_{q^2}, and seconds.

First, we shall assume that $A_r/M_{q^2} = 1$. To justify this, we use estimates similar to the ones in §3.4.8. An integer modulo r can be accommodated in eleven 64-bit words, so we estimate A_r to be 11 clock cycles. Using the carry-less multiplication instruction PCLMULQDQ, a multiplication in $\mathbb{F}_{2^{24}}$ can be performed at a price of approximately 10 clock cycles. This gives us an A_r/M_{q^2} ratio of approximately 1.

Next, since a multiplication in $\mathbb{F}_{2^{24}}$ can be done in approximately 10 clock cycles, we will transform one second on a 2.9 GHz machine (on which the Gröbner bases descent experiments were performed) into $2^{28}M_{q^2}$.

Using these estimates, we see from the third column of Table 3.3 that the overall running time of the new algorithm is approximately $2^{100}M_{q^2}$. As with the case of $\mathbb{F}_{3^{12\cdot 509}}$, the relation generation, continued-fraction descent, classical descent, and Gröbner bases descent steps, and also the relation generation portion of QPA descent, are effectively parallelizable on conventional computers. Moreover, the linear system of equations for finding logarithms of linear polynomials, the 2^{24} linear systems of equations for finding logarithms of irreducible quadratic polynomials, and the 2^{33} linear systems of equations are also effectively parallelizable on conventional computers since each linear system of equations can be expected to be solvable in less than 12 days using a small number of GPUs and CPUs (cf. [74, 14]).

3.5.9 Comparisons with Joux-Lercier

The Joux-Lercier algorithm [81] with pinpointing [76] is an alternative method for computing discrete logarithms in the order-r subgroup of $\mathbb{F}^*_{2^{12\cdot 367}}$. The algorithm works with two polynomial representations of $\mathbb{F}_{2^{12\cdot 367}}$.

The factor base can be taken to be the set of all monic irreducible polynomials of degree at most 4 over $\mathbb{F}_{2^{12}}$ in each of the two representations. The action of the 2^{12}-power Frobenius is used to reduce the factor base size by a factor of 12, yielding a factor base of size $2^{43.4}$. Taking $d_1 = 37$ and $d_2 = 10$ (see Section 2 of [76] for the definitions of d_1 and d_2), the running time of relation generation is approximately $2^{94.0}M_q$, where M_q denotes the cost of a multiplication in $\mathbb{F}_{2^{12}}$ (cf. Section 4 of [76]). The (sparse) matrix in the linear algebra stage has $2^{43.4}$ rows and columns, and approximately 28 nonzero entries per row. Using standard

techniques for solving sparse systems of linear equations [93], the expected cost of the linear algebra is approximately $2^{91.6}$ operations modulo r. Since relation generation is effectively parallelizable, whereas the linear algebra is not amenable to parallelization due to its large size, the dominant step in the Joux-Lercier algorithms is the linear algebra.

In contrast, even though the new algorithm has a greater overall running time of $2^{100}M_{q^2}$, it is effectively parallelizable. Thus a reasonable conclusion is that the new algorithm is more effective than Joux-Lercier for computing logarithms in $\mathbb{F}_{2^{12\cdot 367}}$.

To lend further weight to this conclusion, we observe that special-purpose hardware for solving the relatively-small linear systems of equations in the new algorithm can reasonably be expected to be built at a cost that is well within the budget of a well-funded organization. In 2005, Geiselmann et al. [55] estimated that the cost of special-purpose hardware for solving a linear system where the matrix has 2^{33} rows and columns, and approximately 2^7 nonzero entries (integers modulo 2) per row would be approximately U.S. $2 million; the linear system would be solvable in 2.4 months. For $\mathbb{F}_{2^{12\cdot 367}}$, each matrix in the new algorithm has 2^{24} rows and columns, and approximately 2^{12} nonzero entries (integers modulo r) per row. On the other hand, the cost of special-purpose hardware for solving the linear system encountered in the Joux-Lercier algorithm would be prohibitive.

Our conclusion about the relative weakness of $\mathbb{F}_{2^{12\cdot 367}}$ for discrete logarithm cryptography also applies to the field $\mathbb{F}_{2^{12\cdot 439}}$. Both these conclusions are subject to the caveats in Remark 3.2.

After the estimates in this section were completed, Granger, Kleinjung and Zumbrägel [59, 60] developed several enhancements to the DLP algorithm that substantially decreased its estimated running time. Their improved algorithm was used to compute a discrete logarithm in $\mathbb{F}_{3^{12\cdot 367}}$ in approximately $52,240$ CPU hours.

3.6 Computing discrete logarithms in $\mathbb{F}_{2^{4\cdot 3041}}$

We present a concrete analysis of the DLP algorithm described in §3.3 for computing discrete logarithms in $\mathbb{F}_{2^{4\cdot 3041}}$. In this case, we employ lattices in the classical descent stage [80, 81, 57] (that we describe in §3.6.5), and use Wiedemann's algorithm for performing linear algebra.

The field $\mathbb{F}_{2^{4\cdot 3041}}$ is embedded in its sextic extension $\mathbb{F}_{2^{24\cdot 3041}}$, and it is in the latter field where the DLP algorithm is executed. Thus, we have $q = 2^{12} = 2048$ and $n = 3041$.

As mentioned in §3.1, our main motivation for finding discrete logarithms in $\mathbb{F}_{2^{4\cdot 3041}}$ is to attack the elliptic curve discrete logarithm problem in $E_2(\mathbb{F}_{2^{3041}})$, where E_2 is the supersingular elliptic curve $y^2 + y = x^3 + x$ with $|E_2(\mathbb{F}_{2^{3041}})| = r$ and $r = 2^{3041} - 2^{1521} + 1$ is a 3041-bit prime. The elliptic curve discrete logarithm problem in the order-r subgroup of $E_2(\mathbb{F}_{2^{3041}})$ can be efficiently reduced to the discrete logarithm problem in the order-r subgroup of $\mathbb{F}^*_{2^{4\cdot 3041}}$. We wish to compute $\log_g h \bmod r$, where g is a generator of $\mathbb{F}^*_{2^{4\cdot 3041}}$ and $h \in \mathbb{F}^*_{2^{4\cdot 3041}}$ has order r. Hence, all the linear algebra in the new algorithm is performed modulo the 3041-bit r.

A summary of the parameter choices for the descent is given in Figure 3.3. The cost of each step is given in Table 3.5.

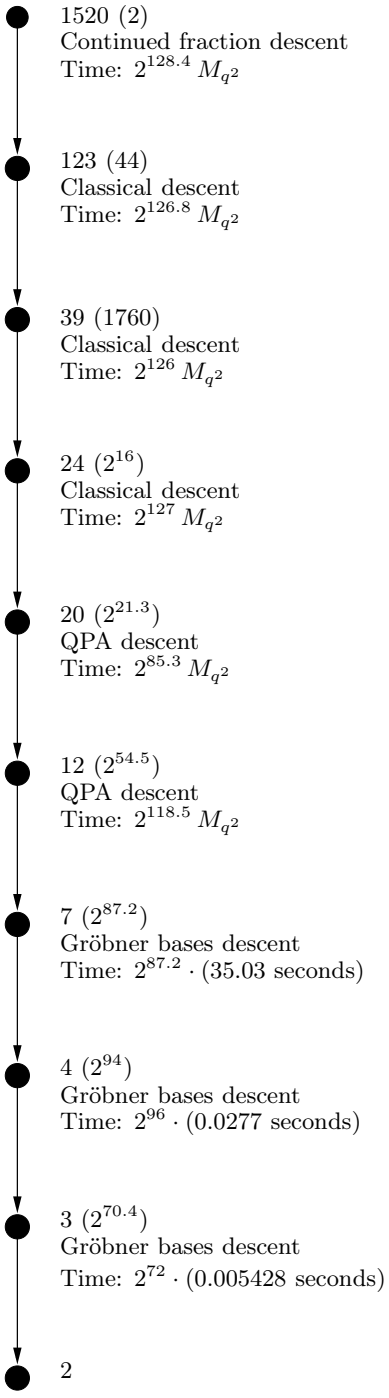

Figure 3.3: A typical path of the descent tree for computing an individual logarithm in $\mathbb{F}_{2^{24\cdot 3041}}$ ($q = 2^{12}$). The numbers in parentheses next to each node are the expected number of nodes at that level. 'Time' is the expected time to generate all nodes at a level.

Finding logarithms of linear polynomials		
Relation generation	$2^{35}M_{q^2}$	$2^{35}M_{q^2}$
Linear algebra	$2^{60}A_r$	$2^{64}M_{q^2}$
Finding logarithms of irreducible quadratic polynomials		
Relation generation	$2^{24}\cdot 2^{44}M_{q^2}$	$2^{68}M_{q^2}$
Linear algebra	$2^{24}\cdot 2^{60}A_r$	$2^{88}M_{q^2}$
Descent		
Continued-fractions (1520 to 123)	$2^{128.4}M_{q^2}$	$2^{128.4}M_{q^2}$
Classical (123 to 39)	$44\cdot 2^{121.3}M_{q^2}$	$2^{126.8}M_{q^2}$
Classical (39 to 24)	$1760\cdot 2^{115}M_{q^2}$	$2^{126}M_{q^2}$
Classical (24 to 20)	$2^{16}\cdot 2^{111}M_{q^2}$	$2^{127}M_{q^2}$
QPA (20 to 12)	$2^{21.3}\cdot(2^{54}M_{q^2}+2^{60}A_r)$	$2^{85.3}M_{q^2}$
QPA (12 to 7)	$2^{54.5}\cdot(2^{52}M_{q^2}+2^{60}A_r)$	$2^{118.5}M_{q^2}$
Gröbner bases (7 to 4)	$2^{87.2}\cdot$(35.03 seconds)	$2^{120.3}M_{q^2}$
Gröbner bases (4 to 3)	$2^{94}\cdot$(0.0277 seconds)	$2^{116.8}M_{q^2}$
Gröbner bases (3 to 2)	$2^{70.4}\cdot$(0.005428 seconds)	$2^{90.9}M_{q^2}$

Table 3.5: Estimated costs of the main steps of the new DLP algorithm for computing discrete logarithms in $\mathbb{F}_{2^{24\cdot 3041}}$ ($q=2^{12}$). A_r and M_{q^2} denote the costs of an addition modulo the 3041-bit prime r and a multiplication in $\mathbb{F}_{2^{24}}$. We use the cost ratio $A_r/M_{q^2}=2^4$, and also assume that 2^{28} multiplications in $\mathbb{F}_{2^{24}}$ can be performed in 1 second (cf. §3.6.8).

3.6.1 Setup

We chose the representations $\mathbb{F}_{2^{12}}=\mathbb{F}_2[U]/(U^{12}+U^7+U^6+U^5+U^3+U+1)$ and $\mathbb{F}_{2^{24}}=\mathbb{F}_{2^{12}}[V]/(V^2+U^{152}V+U^{3307})$. We selected $h_0=(U^{1515}V+U^{3374})X^2+(U^{3690}V+U^{2704})X+(U^{2440}V+U^{142})\in\mathbb{F}_{2^{24}}[X]$ and $h_1=X+U^{2339}V+U^{807}$, and $I_X\in\mathbb{F}_{2^{24}}[X]$ to be the degree-3041 monic irreducible factor of $h_1\cdot X^{2^{12}}-h_0$. The other irreducible factors have degrees 5, 7, 69, 110, 293 and 572.

3.6.2 Finding logarithms of linear polynomials

The factor base $\mathcal{B}_1$ has size 2^{24}. The cost of relation generation is approximately $2^{35}M_{q^2}$, whereas the cost of the linear algebra is approximately $2^{60}A_r$.

3.6.3 Finding logarithms of irreducible quadratic polynomials

For each $u\in\mathbb{F}_{2^{24}}$, the expected cost of computing logarithms of all quadratics in $\mathcal{B}_{2,u}$ is $2^{44}M_{q^2}$ for the computation of $\mathcal{H}(Q)$, and $2^{60}A_r$ for the linear algebra.

3.6.4 Continued-fractions descent

For the continued-fractions descent, we selected $m=123$. The expected cost of this descent is $2^{128.4}M_{q^2}$. The expected number of distinct irreducible factors of degree (at most) 123 obtained is $2A_{2^{24}}(123,1520)\approx 44$.

3.6.5 Classical descent

Let $p = 2$ and $\ell = 12$. Let $s \in [0, \ell]$, and let $R \in \mathbb{F}_{q^2}[X, Y]$ with $\deg_Y R = e$. Then

$$h_1^e \cdot \left[R(X, X^{p^{\ell-s}})\right]^{p^s} = h_1^e \cdot R'(X^{p^s}, X^{p^\ell}) \equiv h_1^e \cdot R'(X^{p^s}, h_0/h_1) \pmod{I_X}, \tag{3.17}$$

where R' is obtained from R by raising all its coefficients to the power p^s.

Let $Q \in \mathbb{F}_{q^2}[X]$ with $\deg Q = D$, and let $m < D$. One selects $s \in [0, \ell]$ and searches for a polynomial $R \in \mathbb{F}_{q^2}[X, Y]$ such that (i) $Q \mid R_1$ where $R_1 = R(X, X^{p^{\ell-s}})$; (ii) $\deg R_1/Q$ and $\deg R_2$ are appropriately balanced where $R_2 = h_1^e \cdot R'(X^{p^s}, h_0/h_1)$; and (iii) both R_1/Q and R_2 are m-smooth. Taking logarithms of both sides of (3.17) then gives an expression for $\log_g Q$ in terms of the logarithms of polynomials of degree at most m.

A family of polynomials R satisfying (i) and (ii) can be constructed by finding a basis $\{(u_1, u_2), (v_1, v_2)\}$ of the lattice

$$L_Q = \{(w_1, w_2) \in \mathbb{F}_{q^2}[X] \times \mathbb{F}_{q^2}[X] \ : \ Q \mid (w_1(X) - w_2(X)X^{p^{\ell-s}})\}$$

where $\deg u_1$, $\deg u_2$, $\deg v_1$, $\deg v_2 \approx D/2$. The points (w_1, w_2) in L_Q can be sampled to obtain polynomials $R(X, Y) = w_1(X) - w_2(X)Y$ satisfying (i) and (ii) by writing

$$(w_1, w_2) = a(u_1, u_2) + b(v_1, v_2) = (au_1 + bv_1, au_2 + bv_2)$$

with $a \in \mathbb{F}_{q^2}[X]$ monic of degree δ and $b \in \mathbb{F}_{q^2}[X]$ of degree $\delta - 1$. We have $\deg w_1, \deg w_2 \approx D/2 + \delta$, so $\deg R_1 = t_1 \approx (D/2 + \delta) + p^{\ell-s}$ and $\deg R_2 = t_2 \approx (D/2 + \delta)p^s + 2$. In order to ensure that the number of lattice points considered is enough to generate a polynomial R such that both R_1/Q and R_2 are m-smooth, the parameters s and δ must be selected so that

$$q^{4\delta} \gg \frac{q^{2(t_1-D)}}{N_{q^2}(m, t_1 - D)} \cdot \frac{q^{2t_2}}{N_{q^2}(m, t_2)}. \tag{3.18}$$

Ignoring the time to compute a balanced basis of L_Q, the expected cost of finding a polynomial R satisfying (i)–(iii) is

$$\frac{q^{2(t_1-D)}}{N_{q^2}(m, t_1 - D)} \cdot \frac{q^{2t_2}}{N_{q^2}(m, t_2)} \cdot \min(S_{q^2}(m, t_1 - D), S_{q^2}(m, t_2)). \tag{3.19}$$

The expected number of distinct irreducible factors of R_1/Q and R_2 is $A_{q^2}(m, t_1 - D) + A_{q^2}(m, t_2)$. In the concrete analysis, we shall assume that each of these irreducible factors has degree exactly m.

Three classical descent stages are employed. In the first stage, we have $D = 123$ and select $m = 39$, $s = 3$, $\delta = 2$, which yield $t_1 = 575$ and $t_2 = 506$. The expected cost of the descent stage for each of the 44 degree-123 polynomials is approximately $2^{93.7} \cdot S_{q^2}(39, 452)$. The expected total number of distinct irreducible polynomials of degree (at most) 39 obtained is approximately 1760.

In the second classical descent stage we have $D = 39$ and select $m = 24$, $s = 4$, $\delta = 2$, which yield $t_1 = 277$ and $t_2 = 338$. The expected cost of the descent for each of the 1760

degree-39 polynomials is approximately $2^{90.2} \cdot S_{q^2}(24, 238)$. The expected total number of distinct irreducible polynomials of degree (at most) 24 obtained is approximately 2^{16}.

In the third classical descent stage we have $D = 24$ and select $m = 20$, $s = 4$, $\delta = 2$, which yield $t_1 = 270$ and $t_2 = 226$. The expected cost of the descent for each of the 2^{16} degree-24 polynomials is approximately $2^{86.9} \cdot S_{q^2}(24, 226)$. The expected total number of distinct irreducible polynomials of degree (at most) 20 obtained is approximately $2^{21.3}$.

3.6.6 QPA descent

Two QPA descent stages are employed. In the first stage, we have $D = 20$ and select $m = 12$. For each Q, the expected cost of relation generation is $2^{34.8} \cdot S_{q^2}(12, 60)$ and the cost of the linear algebra is $2^{60} A_r$. Also for each Q, the expected number of distinct polynomials of degree at most 12 obtained is expected to be $A_{q^2}(12, 60) \cdot q^2 \approx 2^{33.2}$. Thus, the total number of distinct polynomials of degree at most 12 obtained after the first QPA descent stage is approximately $2^{54.5}$.

In the second stage, we have $D = 12$ and select $m = 7$. For each Q, the expected cost of relation generation is $2^{35} \cdot S_{q^2}(7, 36)$ and the cost of the linear algebra is $2^{60} A_r$. Also for each Q, the expected number of distinct polynomials of degree at most 7 obtained is expected to be $A_{q^2}(7, 36) \cdot q^2 \approx 2^{32.7}$. Thus, the total number of distinct polynomials of degree at most 7 obtained after the second QPA descent stage is approximately $2^{87.2}$.

3.6.7 Gröbner bases descent

Three Gröbner bases descent stages are employed. The first stage has $D = 7$ and $m = 4$, and is expected to yield approximately $2^{99.2}$ polynomials of degree (at most) 4. The second stage has $D = 4$ and $m = 3$ and is applied to all the 2^{94} monic irreducible quartics over $\mathbb{F}_{2^{24}}$. The third stage has $D = 3$ and $m = 2$ and is applied to all the $2^{70.4}$ monic irreducible cubics over $\mathbb{F}_{2^{24}}$. Our experiments were run on a 2.9 GHz Intel core i7-3520M using Magma's implementation of Faugère's F4 algorithm [47].

In the first stage, for each degree-7 polynomial Q we have to solve a system of 26 quadratic polynomial equations in 32 variables over $\mathbb{F}_q$ (cf. (3.13)). After fixing some variables, each degree-5 R obtained from the variety of the resulting ideal is tested for 4-smoothness. If no 4-smooth R is obtained, we randomly fix some other subset of variables and repeat. We ran 10,000 Gröbner bases descent experiments with randomly-selected degree-7 polynomials Q. On average, we had to find 1.806 Gröbner bases for each Q. The average number of R's tested for 4-smoothness for each Q was 1.252. The average time spent on each Q was 35.03 seconds.

For the second and third stages, we use the experimental results from §3.5.7.

3.6.8 Overall running time

The second column of Table 4.2 gives the running time estimates for the main steps of the new DLP algorithm in three units of time: A_r, M_{q^2}, and seconds. In order to assess the overall time, we make some assumptions about the ratios of these units of time.

First, we shall assume that $A_r/M_{q^2} = 2^4$. To justify this, we observe that a 3041-bit integer can be stored in 48 64-bit words. The cost of A_r can be estimated to be 48 clock

cycles. Using the carry-less multiplication instruction PCLMULQDQ, a multiplication in $\mathbb{F}_{2^{24}}$ can be performed at a price of 3-4 clock cycles. This gives us an A_r/M_{q^2} ratio of approximately 2^4.

Next, since a multiplication in M_{q^2} can be done in 15 clock cycles, we will transform one second on a 2.9 GHz machine (on which the Gröbner bases descent experiments were performed) into $2^{28}M_{q^2}$.

Using these estimates, we see from the third column of Table 3.5 that the overall running time of the new algorithm is approximately $2^{129.3}M_{q^2}$. The new algorithm is effectively parallelizable, since each linear system of equations can be expected to be solvable in less than 12 days using a small number of GPUs and CPUs.

3.6.9 Comparisons

The upper bound of $2^{129.3}M_{q^2}$ on the running time of the new algorithm for computing logarithms in $\mathbb{F}_{2^{4\cdot 3041}}$ convincingly demonstrates that this field offers drastically less security than the 2^{192} resistance to attacks by Coppersmith's algorithm [40, 94]. This decrease in security is even more pronounced when one considers that Coppersmith's algorithm is not parallelizable whereas the new algorithm is effectively parallelizable.

3.7 Concluding remarks

Our concrete analysis of the new algorithm of Joux and Barbulescu et al. has shown that the supersingular elliptic curve over $\mathbb{F}_{3^{509}}$ with embedding degree 6 is significantly less resistant to attacks on the elliptic curve discrete logarithm problem than previously believed. Consequently, this elliptic curve is not suitable for implementing pairing-based cryptosystems. Our analysis applies equally well to the supersingular elliptic curve over $\mathbb{F}_{3^{5\cdot 97}}$ with embedding degree 6 that has been proposed for compact hardware implementation of pairing-based cryptosystems by Estibals [46]. Moreover, the new algorithm appears advantageous over Joux-Lercier's and Coppersmith's algorithms for discrete logarithm computations in $\mathbb{F}_{2^{12\cdot 367}}$ with embedding degree 12 and $\mathbb{F}_{2^{4\cdot 3041}}$ with embedding degree 4.

An important open question is whether the new algorithm or its implementation can be improved to the extent that the discrete logarithm problem in $\mathbb{F}_{3^{6\cdot 509}}$ can be feasibly solved using existing computer technology.

4 Weakness of $\mathbb{F}_{3^{6\cdot 1429}}$ and Discrete Logarithm Computations in $\mathbb{F}_{3^{6\cdot 137}}$ and $\mathbb{F}_{3^{6\cdot 163}}$

4.1 Introduction

The setup in the new algorithm of Joux and Barbulescu et al. described in Chapter 3 imposes some restrictions on the algorithm parameters which limits the range of fields on which the algorithm is effective. Suppose that one wishes to compute logarithms in $\mathbb{F}_{q^{2n}}$,[1] where q is the power of a small prime and n is prime. The new algorithm of Joux and Barbulescu et al. represents $\mathbb{F}_{q^{2n}}$ as $\mathbb{F}_{q^2}[X]/(I_X)$, where I_X is a degree-n irreducible factor of $h_1X^q - h_0$ in $\mathbb{F}_{q^2}[X]$, and $h_0, h_1 \in \mathbb{F}_{q^2}[X]$ have small degree (say, 2); hence, one must have $n \leq q+2$. For example, logarithms in $\mathbb{F}_{3^{6\cdot 509}}$ can be computed by first embedding the field in the quadratic extension $\mathbb{F}_{(3^6)^{2\cdot 509}}$; one can take $q = 3^6 = 729$ and $n = 509$. However, if one wishes to compute logarithms in $\mathbb{F}_{3^{6\cdot 1429}}$, then the smallest extension that meets the setup criteria is $\mathbb{F}_{(3^9)^{2\cdot 1429}}$; this field is too large for the new attacks to be effective.

At the ECC 2013 conference, Granger and Zumbrägel [64] (see also [60]) presented a modification of the new algorithms that alleviates the aforementioned restrictions. Their idea is to select I_X as a degree-n irreducible factor of $h_1(X^q) \cdot X - h_0(X^q)$, where $h_0, h_1 \in \mathbb{F}_{q^2}[X]$ have small degree (say, 2); the condition on q and n is then relaxed to $n \leq 2q+1$. While this modification does not affect the asymptotic run time of the new algorithms, it is very successful in increasing the effectiveness of the new algorithms in practice. Granger, Kleinjung and Zumbrägel [60] presented several enhancements to the algorithms and computed logarithms in the 4404-bit field $\mathbb{F}_{2^{12\cdot 367}}$ in approximately 52,240 CPU hours. Furthermore, they drastically reduced the estimated time to compute logarithms in the 4892-bit field $\mathbb{F}_{2^{4\cdot 1223}}$ to 2^{59} modular multiplications. Note that the fields $\mathbb{F}_{2^{12\cdot 367}}$ and $\mathbb{F}_{2^{4\cdot 1223}}$ offer approximately 128 bits of security against attacks by Coppersmith's algorithm.

The purpose of this chapter, in joint work with A. Menezes, T. Oliveira and F. Rodríguez-Henríquez [4, 5], is to show that the new algorithm of Joux and Barbulescu et al. can have a more drastic impact on the security of the supersingular elliptic curves $y^2 = x^3 - x \pm 1$ over $\mathbb{F}_{3^n}$ when the Granger-Zumbrägel polynomial representation is used. Recall that the supersingular elliptic curves $y^2 = x^3 - x \pm 1$ over $\mathbb{F}_{3^n}$ have embedding degree 6 and were proposed for Type 1 pairing-based cryptography in several early papers on pairing-based cryptography [31, 18, 52, 62].

At high security levels, we consider the embedding degree-6 elliptic curve $E : y^2 =$

[1] In general, one wishes to compute logarithms in $\mathbb{F}_{p^{cn}}$ where p is a small prime, n a prime and c a small integer. To accomplish this, one embeds $\mathbb{F}_{p^{cn}}$ in $\mathbb{F}_{(p^\ell)^{kn}}$ where $k > 1$ and $c \mid \ell k$.

$x^3 - x - 1$ over $\mathbb{F}_{3^{1429}}$. We have $|E(\mathbb{F}_{3^{1429}})| = cr$ where r is a 2223-bit prime and c is a 43-bit cofactor. The finite field $\mathbb{F}_{3^{6\cdot 1429}}$ offers approximately 192 bits of security against attacks on the DLP by Coppersmith's algorithm. In contrast, our concrete analysis shows that the order-r subgroup of the multiplicative group of this field offers at most 96 bits of security against the new attack with the Granger-Zumbrägel polynomial representation.

In addition, we demonstrate that, with modest computational resources, the new algorithm can be used to solve instances of the discrete logarithm problem that remain beyond the reach of classical algorithms. The first target field is the 1303-bit field $\mathbb{F}_{3^{6\cdot 137}}$; this field does not enjoy any Kummer-like properties. More precisely, we are interested in solving the discrete logarithm problem in the order-r subgroup $\mathbb{G}$ of $\mathbb{F}^*_{3^{6\cdot 137}}$, where $r = (3^{137} - 3^{69} + 1)/7011427850892440647$ is a 155-bit prime. The discrete logarithm problem in this group is of cryptographic interest because the values of the bilinear pairing derived from the supersingular elliptic curve $E : y^2 = x^3 - x + 1$ over $\mathbb{F}_{3^{137}}$ lie in $\mathbb{G}$. Consequently, if logarithms in $\mathbb{G}$ can be computed efficiently then the associated bilinear pairing is rendered cryptographically insecure. Note that since r is a 155-bit prime, Pollard's rho algorithm [109] for computing logarithms in $\mathbb{G}$ is infeasible. Moreover, recent work on computing logarithms in the 809-bit field $\mathbb{F}_{2^{809}}$ [14] suggests that Coppersmith's algorithm is infeasible for computing logarithms in $\mathbb{G}$, whereas recent work on computing logarithms in the 923-bit field $\mathbb{F}_{3^{6\cdot 97}}$ [69] (see also [115]) indicates that computing logarithms in $\mathbb{G}$ using the Joux-Lercier algorithm [81] would be a formidable challenge. In contrast, we show that the new algorithm can be used to compute logarithms in $\mathbb{G}$ in just a few days using a small number of CPUs; more precisely, our computation consumed a total of 888 CPU hours. The computational effort expended in our experiment is relatively small, despite the fact that our implementation was done using the computer algebra system Magma V2.20-2 [98] and is far from optimal (see Remark 3.3).

The second target field is the 1551-bit field $\mathbb{F}_{3^{6\cdot 163}}$; this field does not enjoy any Kummer-like properties. More precisely, we are interested in solving the discrete logarithm problem in the order-r subgroup $\mathbb{G}$ of $\mathbb{F}^*_{3^{6\cdot 163}}$, where $r = 3^{163} + 3^{82} + 1$ is a 259-bit prime. The discrete logarithm problem in this group is of cryptographic interest because the values of the bilinear pairing derived from the supersingular elliptic curve $E : y^2 = x^3 - x - 1$ over $\mathbb{F}_{3^{163}}$ lie in $\mathbb{G}$. This bilinear pairing was first considered by Boneh, Lynn and Shacham in their landmark paper on short signature schemes [31]; see also [63]. Furthermore, the bilinear pairing derived from the quadratic twist of E was one of the pairings implemented by Galbraith, Harrison and Soldera [52]. Again, we show that the new algorithm can be used to compute logarithms in $\mathbb{G}$ in just a few days using a small number of CPUs; our computation used 1201 CPU hours.

The remainder of the chapter is organized as follows. In §4.2 we review the DLP algorithms of Joux and Barbulescu et al. while incorporating the Granger-Zumbrägel polynomial representation. Our concrete analysis for $\mathbb{F}_{3^{6\cdot 1429}}$ is then presented in §4.3. Our experimental results with computing logarithms in $\mathbb{F}_{3^{6\cdot 137}}$ and $\mathbb{F}_{3^{6\cdot 163}}$ are reported in §4.4 and §4.5, respectively. We make some concluding remarks in §4.6.

4.2 The DLP algorithm of Joux, Barbulescu et al. and Granger-Zumbrägel

The DLP algorithm we describe is due to Joux [77], with a descent step from the quasi-polynomial time algorithm of Barbulescu et al. [16], and a polynomial representation (selection of h_0 and h_1) due to Granger and Zumbrägel [64] (see also [60]). The description of the algorithm closely follows the description in Chapter 3; the most important changes are the incorporation of the polynomial selection of Granger and Zumbrägel and the use of lattices in the classical descent stage [80, 81, 57].

Let $\mathbb{F}_{q^{kn}}$ be a finite field where $n \leq 2q+1$. The elements of $\mathbb{F}_{q^{kn}}$ are represented as polynomials of degree at most $n-1$ over $\mathbb{F}_{q^k}$. Let $N = q^{kn}-1$. Let g be an element of order N in $\mathbb{F}^*_{q^{kn}}$, and let $h \in \mathbb{F}^*_{q^{kn}}$. We wish to compute $\log_g h$. The algorithm proceeds by first finding the logarithms of all degree-one (§4.2.2) and degree-two (§4.2.3) elements in $\mathbb{F}_{q^{kn}}$. Then, in the descent stage, $\log_g h$ is expressed as a linear combination of logarithms of degree-one and degree-two $\mathbb{F}_{q^{kn}}$ elements. The descent stage proceeds in several steps, each expressing the logarithm of a degree-D element as a linear combination of the logarithms of elements of degree $\leq m$ for some $m < D$. Four descent methods are used; these are described in §4.2.4–§4.2.7. The cost of each step is given in Table 4.1.

Notation. For $\gamma \in \mathbb{F}_{q^k}$, $\overline{\gamma}$ denotes the element $\gamma^{q^{k-1}}$. For $P \in \mathbb{F}_{q^k}[X]$, $\overline{P}$ denotes the polynomial obtained by raising each coefficient of P to the power q^{k-1}. The cost of an integer addition modulo N is denoted by A_N, and the cost of a multiplication in $\mathbb{F}_{q^k}$ is denoted by M_{q^k}.

The projective general linear group of degree 2 over $\mathbb{F}_q$ is denoted $\mathrm{PGL}_2(\mathbb{F}_q)$. $\mathcal{P}_{q,k}$ is a set of distinct representatives of the left cosets of $\mathrm{PGL}_2(\mathbb{F}_q)$ in $\mathrm{PGL}_2(\mathbb{F}_{q^k})$; note that

$$|\mathcal{P}_{q,k}| = \frac{q^{3k}-q^k}{q^3-q} = \sum_{i=1}^{\lfloor \frac{3}{2}k \rfloor - 1} q^{3k-2i-1}.$$

A matrix $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in \mathcal{P}_{q,k}$ is identified with the quadruple (a,b,c,d).

4.2.1 Setup

Select polynomials $h_0, h_1 \in \mathbb{F}_{q^k}[X]$ of small degree so that

$$X \cdot h_1(X^q) - h_0(X^q) \tag{4.1}$$

has a irreducible factor I_X of degree n in $\mathbb{F}_{q^k}[X]$; we will henceforth assume that $\max(\deg h_0, \deg h_1) = 2$. Note that

$$X \equiv \frac{h_0(X^q)}{h_1(X^q)} \equiv \left(\frac{\overline{h}_0(X)}{\overline{h}_1(X)}\right)^q \pmod{I_X} \tag{4.2}$$

The field $\mathbb{F}_{q^{kn}}$ is represented as $\mathbb{F}_{q^{kn}} = \mathbb{F}_{q^k}[X]/(I_X)$ and the elements of $\mathbb{F}_{q^{kn}}$ are represented as polynomials in $\mathbb{F}_{q^k}[X]$ of degree at most $n-1$. Let g be a generator of $\mathbb{F}^*_{q^{kn}}$.

Finding logarithms of linear polynomials (§4.2.2)	
Relation generation	$6q^k \cdot S_{q^k}(1,3)$
Linear algebra	$q^{2k+1} \cdot A_N$
Finding logarithms of irreducible quadratic polynomials (§4.2.3)	
Relation generation	$\frac{q^{8k}}{N_{q^k}(1,6)} \cdot S_{q^k}(1,6)$
Linear algebra	$q^{3k+1} \cdot A_N$
Descent (Degree D to degree m)	
Continued-fractions (§4.2.4)	$\{D = n-1\} \left(\frac{q^{n-1}}{N_{q^k}(m,(n-1)/2)}\right)^2 \cdot S_{q^k}(m,(n-1)/2)$
Classical (§4.2.5)	$\frac{q^{2(t_1-D+t_2)}}{N_{q^k}(m,t_1-D)N_{q^k}(m,t_2)} \cdot \min(S_{q^k}(m,t_1-D), S_{q^k}(m,t_2))$ or
	$\frac{q^{2(t_1+t_2-D)}}{N_{q^k}(m,t_1)N_{q^k}(m,t_2-D)} \cdot \min(S_{q^k}(m,t_1), S_{q^k}(m,t_2-D))$
QPA (§4.2.6)	$\frac{q^{6D+2}}{N_{q^k}(m,3D)} \cdot S_{q^k}(m,3D) + q^5 \cdot A_N$
Gröbner bases (§4.2.7)	$G_{q^k}(m,D) + \frac{q^{6m-2D}}{N_{q^k}(m,3m-D)} \cdot S_{q^k}(m,3m-D)$

Table 4.1: Estimated costs of the main steps of the useful DLP algorithm for computing discrete logarithms in $\mathbb{F}_{q^{kn}}$. A_N and M_{q^k} denote the costs of an addition modulo N and a multiplication in $\mathbb{F}_{q^k}$. See §3.3.5 for the definitions of t_1 and t_2. The Gröbner basis cost $G_{q^k}(m,D)$ is defined in §4.2.7.

4.2.2 Finding logarithms of linear polynomials

Let $\mathcal{B}_1 = \{X + a \mid a \in \mathbb{F}_{q^k}\}$, and note that $|\mathcal{B}_1| = q^k$. To compute the logarithms of $\mathcal{B}_1$-elements, we first generate linear relations of these logarithms. Let $(a,b,c,d) \in \mathcal{P}_{q,k}$. Substituting $Y \mapsto (aX+b)/(cX+d)$ into the systematic equation

$$Y^q - Y = \prod_{\alpha \in \mathbb{F}_q} (Y - \alpha) \tag{4.3}$$

and then multiplying by $(cX+d)^{q+1}$ yields

$$\begin{aligned}(aX+b)^q(cX+d) - (aX+b)(cX+d)^q \\ = (cX+d) \cdot \prod_{\alpha \in \mathbb{F}_q} [(a-\alpha c)X + (b - \alpha d)].\end{aligned} \tag{4.4}$$

Replacing X by $(\overline{h}_0/\overline{h}_1)^q$ in the linear terms $aX+b$ and $cX+d$ occurring in the left side of (4.4) and then clearing denominators yields

$$\left((aX+b)(\overline{c}\,\overline{h}_0+\overline{d}\,\overline{h}_1)-(\overline{a}\,\overline{h}_0+\overline{b}\,\overline{h}_1)(cX+d)\right)^q \equiv \overline{h}_1^q\cdot(cX+d)\cdot\prod_{\alpha\in\mathbb{F}_q}[(a-\alpha c)X+(b-\alpha d)]. \tag{4.5}$$

If the polynomial on the left side of (4.5) is 1-smooth, then taking logarithms of both sides of (4.5) yields a linear relation of the logarithms of $\mathcal{B}_1$-elements and the logarithm of $\overline{h}_1$. The probability that the left side of (4.5) is 1-smooth is $N_{q^k}(1,3)/q^6 \approx \frac{1}{6}$. Thus, after approximately $6q^k$ trials one expects to obtain (slightly more than) q^k relations. The cost of the relation generation stage is $6q^k \cdot S_{q^k}(1,3)$. The logarithms can then be obtained by using Wiedemann's algorithm for solving sparse systems of linear equations [124, 41]. The expected cost of the linear algebra is $q^{2k+1}\cdot A_N$ since each equation has approximately q nonzero terms.

4.2.3 Finding logarithms of irreducible quadratic polynomials

Let $u\in\mathbb{F}_{q^k}$, and let $Q(X)=X^2+uX+v\in\mathbb{F}_{q^k}[X]$ be an irreducible quadratic. Define $\mathcal{B}_{2,u}$ to be the set of all irreducible quadratics of the form X^2+uX+w in $\mathbb{F}_{q^k}[X]$; one expects that $|\mathcal{B}_{2,u}|\approx(q^k-1)/2$. The logarithms of all elements in $\mathcal{B}_{2,u}$ are found simultaneously using one application of QPA descent (see §3.3.6). More precisely, one first collects relations of the form (4.14), where the left side of (4.14) factors as a product of linear polynomials (whose logarithms are known). The expected number of relations one can obtain is $|\mathcal{P}_{q,k}|\cdot N_{q^k}(1,6)/q^{6k}$. Provided that this number is significantly greater than $|\mathcal{B}_{2,u}|$, the matrix $\mathcal{H}(Q)$ is expected to have full (column) rank. One can then solve the resulting system of linear equations to obtain the logarithms of all irreducible translates $Q+w$ of Q. This step is repeated for each $u\in\mathbb{F}_{q^k}$. Hence, there are q^k independent linear systems of equations to be solved.

For each $u\in\mathbb{F}_{q^k}$, the cost of the relation generation is $q^{7k}/N_{q^k}(1,6)\cdot S_{q^k}(1,6)$, while the linear algebra cost is $q^{2k+1}\cdot A_N$.

4.2.4 Continued-fractions descent

Recall that we wish to compute $\log_g h$, where $h\in\mathbb{F}_{q^{kn}}=\mathbb{F}_{q^k}[X]/(I_X)$. We will henceforth assume that $\deg h=n-1$. The descent stage begins by multiplying h by a random power of g. The extended Euclidean algorithm is used to express the resulting field element h' in the form $h'=w_1/w_2$ where $\deg w_1,\deg w_2\approx n/2$ [28]; for simplicity, we shall assume that n is odd and $\deg w_1=\deg w_2=(n-1)/2$. This process is repeated until both w_1 and w_2 are m-smooth for some chosen $m<(n-1)/2$. This gives $\log_g h'$ as a linear combination of logarithms of polynomials of degree at most m. The expected cost of this continued-fractions descent step is approximately

$$\left(\frac{(q^k)^{(n-1)/2}}{N_{q^k}(m,(n-1)/2)}\right)^2\cdot S_{q^k}(m,(n-1)/2). \tag{4.6}$$

The expected number of distinct irreducible factors of w_1 and w_2 is $2A_{q^k}(m,(n-1)/2)$. In the concrete analysis, we shall assume that each of these irreducible factors has degree exactly m. The logarithm of each of these degree-m polynomials is then expressed as a linear combination of logarithms of smaller degree polynomials using one of the descent methods described in §4.2.5, §4.2.6 and §4.2.7.

4.2.5 Classical descent

Let p be the characteristic of $\mathbb{F}_q$, and let $q = p^\ell$. Let $s \in [0,\ell]$, and let $R \in \mathbb{F}_{q^k}[X,Y]$. Then

$$\left[R\left(X,(\overline{h}_0/\overline{h}_1)^{p^{\ell-s}}\right)\right]^{p^s} = R'\left(X^{p^s},\overline{h}_0^{p^\ell}\right) = R'\left(X^{p^s},h_0\left(X^{p^\ell}\right)\right) \equiv R'\left(X^{p^s},X\right) \pmod{I_X} \tag{4.7}$$

where R' is obtained from R by raising all its coefficients to the power p^s. Let $\mu = \deg_Y R$. Then multiplying both sides of (4.7) by $\overline{h}_1^{q\mu}$ gives Hence

$$\left[\overline{h}_1^{p^{\ell-s}\cdot\mu}\cdot R\left(X,(\overline{h}_0/\overline{h}_1)^{p^{\ell-s}}\right)\right]^{p^s} \equiv \overline{h}_1^{q\mu}\cdot R'(X^{p^s},X) \pmod{I_X}. \tag{4.8}$$

Let $Q \in \mathbb{F}_{q^k}[X]$ with $\deg Q = D$, and let $m < D$. In the Joux-Lercier descent method [81], as modified by Göloğlu et al. [57], one selects $s \in [0,\ell]$ and searches for a polynomial $R \in \mathbb{F}_{q^k}[X,Y]$ such that (i) $Q \mid R_1$ where $R_1 = \overline{h}_1^{p^{\ell-s}\mu}R(X,(\overline{h}_0/\overline{h}_1)^{p^{\ell-s}})$; (ii) $\deg R_1/Q$ and $\deg R_2$ are appropriately balanced where $R_2 = R'(X^{p^s},X)$; and (iii) both R_1/Q and R_2 are m-smooth. Taking logarithms of both sides of (4.8) then gives an expression for $\log_g Q$ in terms of the logarithms of polynomials of degree at most m.

A family of polynomials R satisfying (i) and (ii) can be constructed by finding a basis $\{(u_1,u_2),$
$(v_1,v_2)\}$ of the lattice

$$L_Q = \{(w_1,w_2) \in \mathbb{F}_{q^k}[X]\times\mathbb{F}_{q^k}[X] \;:\; Q \mid (w_1(X)\overline{h}_1(X)^{p^{\ell-s}} - w_2(X)\overline{h}_0(X)^{p^{\ell-s}})\}$$

where $\deg u_1$, $\deg u_2$, $\deg v_1$, $\deg v_2 \approx D/2$. The points (w_1,w_2) in L_Q can be sampled to obtain polynomials $R(X,Y) = w_1(X) - w_2(X)Y$ satisfying (i) and (ii) by writing

$$(w_1,w_2) = a(u_1,u_2) + b(v_1,v_2) = (au_1+bv_1, au_2+bv_2)$$

with $a \in \mathbb{F}_{q^k}[X]$ monic of degree δ and $b \in \mathbb{F}_{q^k}[X]$ of degree $\delta-1$. The number of lattice points to consider is therefore $(q^k)^{2\delta}$. We have $\deg w_1, \deg w_2 \approx D/2+\delta$, so $\deg R_1 = t_1 \approx (D/2+\delta)+2p^{\ell-s}$ and $\deg R_2 = t_2 \approx (D/2+\delta)p^s+1$. In order to ensure that there are sufficiently many such lattice points to generate a polynomial R for which both R_1/Q and R_2 are m-smooth, the parameters s and δ must be selected so that

$$q^{2k\delta} \gg \frac{q^{k(t_1-D)}}{N_{q^k}(m,t_1-D)}\cdot\frac{q^{kt_2}}{N_{q^k}(m,t_2)}. \tag{4.9}$$

Ignoring the time to compute a balanced basis of L_Q, the expected cost of finding a polynomial R satisfying (i)–(iii) is

$$\frac{q^{k(t_1-D)}}{N_{q^k}(m,t_1-D)}\cdot\frac{q^{kt_2}}{N_{q^k}(m,t_2)}\cdot\min(S_{q^k}(m,t_1-D),S_{q^k}(m,t_2)). \tag{4.10}$$

The expected number of distinct irreducible factors of R_1/Q and R_2 is $A_{q^k}(m, t_1 - D) + A_{q^k}(m, t_2)$. In the concrete analysis, we shall assume that each of these irreducible factors has degree exactly m.

An alternative to the above method is to select $s \in [0, \ell]$ and search for $R \in \mathbb{F}_{q^k}[X, Y]$ such that (i) $Q \mid R_2$; (ii) $\deg R_1$ and $\deg R_2/Q$ are appropriately balanced; and (iii) both R_1 and R_2/Q are m-smooth. A family of polynomials R satisfying (i) and (ii) can be constructed by finding a basis $\{(u_1, u_2), (v_1, v_2)\}$ of the lattice

$$L_Q = \{(w_1, w_2) \in \mathbb{F}_{q^k}[X] \times \mathbb{F}_{q^k}[X] \ : \ Q \mid (w_1(X) - w_2(X)X^{p^s})\}$$

where $\deg u_1$, $\deg u_2$, $\deg v_1$, $\deg v_2 \approx D/2$. The points (w_1, w_2) in L_Q can be sampled as before to obtain polynomials $R(X, Y) = w_1''(Y) - w_2''(Y)X$ satisfying (i) and (ii) where w'' is obtained from w by raising all its coefficients to the power p^{-s}. We have $\deg w_1, \deg w_2 \approx D/2 + \delta$, so $\deg R_1 = t_1 \approx 2(D/2 + \delta)p^{\ell - s} + 1$ and $\deg R_2 = t_2 \approx (D/2 + \delta) + p^s$. In order to ensure that there are sufficiently many such lattice points to generate a polynomial R for which both R_1 and R_2/Q are m-smooth, the parameters s and δ must be selected so that

$$q^{2k\delta} \gg \frac{q^{kt_1}}{N_{q^k}(m, t_1)} \cdot \frac{q^{k(t_2 - D)}}{N_{q^k}(m, t_2 - D)}. \tag{4.11}$$

Ignoring the time to compute a balanced basis of L_Q, the expected cost of finding a polynomial R satisfying (i)–(iii) is

$$\frac{q^{kt_1}}{N_{q^k}(m, t_1)} \cdot \frac{q^{k(t_2 - D)}}{N_{q^k}(m, t_2 - D)} \cdot \min(S_{q^k}(m, t_1), S_{q^k}(m, t_2 - D)). \tag{4.12}$$

The expected number of distinct irreducible factors of R_1, R_2/Q is $A_{q^k}(m, t_1) + A_{q^k}(m, t_2 - D)$.

4.2.6 QPA descent

Let $Q \in \mathbb{F}_{q^k}[X]$ with $\deg Q = D$, and let $m \in [\lceil D/2 \rceil, D - 1]$. Let $(a, b, c, d) \in \mathcal{P}_{q,k}$. Substituting $Y \mapsto (aQ + b)/(cQ + d)$ into the systematic equation (4.3) and multiplying by $(cQ + d)^{q+1}$ yields

$$(aQ + b)^q(cQ + d) - (aQ + b)(cQ + d)^q = (cQ + d) \prod_{\alpha \in \mathbb{F}_q} [(a - \alpha c)Q + (b - \alpha d)]. \tag{4.13}$$

Noticing that

$$cQ(X) + d \equiv cQ\left(\left(\frac{\overline{h}_0}{\overline{h}_1}\right)^q\right) + d \equiv \left(\overline{c}\,\overline{Q}\left(\frac{\overline{h}_0}{\overline{h}_1}\right) + \overline{d}\right)^q \equiv \overline{h}_1^{-Dq}\left(\overline{c}\,\widetilde{Q} + \overline{d}\,\overline{h}_1^D\right)^q \pmod{I_X}$$

where $\widetilde{Q}(X) = \overline{h}_1^D \cdot \overline{Q}(\overline{h}_0/\overline{h}_1)$, we obtain

$$\left((aQ + b)(\overline{c}\widetilde{Q} + \overline{d}\,\overline{h}_1^D) - (\overline{a}\widetilde{Q} + \overline{b}\,\overline{h}_1^D)(cQ + d)\right)^q \tag{4.14}$$
$$\equiv \overline{h}_1^{Dq} \cdot (cQ + d) \cdot \prod_{\alpha \in \mathbb{F}_q} [(a - \alpha c)Q + (b - \alpha d)] \pmod{I_X}.$$

Note that the polynomial within the main parentheses on the left side of (4.14) has degree $\leq 3D$. If this polynomial is m-smooth, then (4.14) yields a linear relation of the logarithms of some degree-m polynomials and logarithms of translates of Q. After collecting slightly more than q^k such relations, one searches for a linear combination of these relations that eliminates all translates of Q except for Q itself. To achieve this, consider vectors in $(\mathbb{Z}_N)^{q^k}$ with coordinates indexed by elements $\lambda \in \mathbb{F}_{q^k}$. For each relation, we define a vector v whose entry v_λ is 1 if $Q - \lambda$ appears in the right side of (4.14), and 0 otherwise. If the resulting matrix $\mathcal{H}(Q)$ of row vectors has full column rank, then one obtains an expression for $\log_g Q$ in terms of the logarithms of polynomials of degree $\leq m$. The number of distinct polynomials of degree $\leq m$ in this expression is expected to be $A_{q^k}(m, 3D) \cdot q^k$; in the concrete analysis we shall assume that each of these polynomials has degree exactly m.

Since the probability that a degree-$3D$ polynomial is m-smooth is $N_{q^k}(m, 3D)/(q^k)^{3D}$, one must have

$$\frac{N_{q^k}(m, 3D)}{q^{2kD}} \cdot |\mathcal{P}_{q,k}| \gg q^k \tag{4.15}$$

in order to ensure that $\mathcal{H}(Q)$ has $\gg q^k$ rows, whereby $\mathcal{H}(Q)$ can be expected to have full rank.

The expected cost of the relation generation portion of QPA descent is $q^{k(3D+1)} \cdot S_{q^k}(m, 3D)/N_{q^k}(m, 3D)$, while the cost of the linear algebra is $q^{2k+1} \cdot A_N$.

4.2.7 Gröbner bases descent

Let $Q \in \mathbb{F}_{q^k}[X]$ with $\deg Q = D$, and let $m < D$. In Joux's new descent method [77, §5.3], one finds degree-m polynomials $k_1, k_2 \in \mathbb{F}_{q^k}[X]$ such that $Q \mid G$, where

$$G = (k_1 \widetilde{k}_2 - \widetilde{k}_1 k_2) \bmod I_X$$

and where $\widetilde{k}_1 = \overline{h}_1^m \overline{k}_1(\overline{h}_0/\overline{h}_1)$ and $\widetilde{k}_2 = \overline{h}_1^m \overline{k}_2(\overline{h}_0/\overline{h}_1)$. We then have

$$\overline{h}_1^{mq} \cdot k_2 \cdot \prod_{\alpha \in \mathbb{F}_q} (k_1 - \alpha k_2) \equiv G(X)^q \pmod{I_X}$$

as can be seen by making the substitution $Y \mapsto k_1/k_2$ into the systematic equation (4.3) and clearing denominators. Note that $\deg(\widetilde{k}_1) = \deg(\widetilde{k}_2) = 2m$. Hence, if $3m < n$ then $G = k_1\widetilde{k}_2 - \widetilde{k}_1 k_2$ and so $G(X) = Q(X)R(X)$ for some $R \in \mathbb{F}_{q^k}[X]$ with $\deg R = 3m - D$. If R is m-smooth, we obtain a linear relationship between $\log_g Q$ and logs of degree-m polynomials by taking logarithms of both sides of the following:

$$\overline{h}_1^{mq} \cdot k_2 \cdot \prod_{\alpha \in \mathbb{F}_q} (k_1 - \alpha k_2) \equiv (Q(X)R(X))^q \pmod{I_X}. \tag{4.16}$$

To determine (k_1, k_2, R) that satisfy

$$k_1 \widetilde{k}_2 - \widetilde{k}_1 k_2 = Q(X)R(X), \tag{4.17}$$

one can transform (4.17) into a system of multivariate quadratic equations over $\mathbb{F}_q$. Specifically, each coefficient of k_1, k_2 and R is written using k variables over $\mathbb{F}_q$. The coefficients of $\widetilde{k}_1$ and $\widetilde{k}_2$ can then be written in terms of the coefficients of k_1 and k_2. Hence, equating coefficients of X^i of both sides of (4.17) yields $3m+1$ quadratic equations. Equating $\mathbb{F}_q$-components of these equations then yields $k(3m+1)$ bilinear equations in $k(5m-D+3)$ variables over $\mathbb{F}_q$. This system of equations can be solved by finding a Gröbner basis for the ideal it generates. Finally, solutions (k_1, k_2, R) are tested until one is found for which R is m-smooth. This yields an expression for $\log_g Q$ in terms of the logarithms of approximately $q+1+A_{q^k}(m, 3m-D)$ polynomials of degree (at most) m; in the analysis we shall assume that each of the polynomials has degree exactly m.

Denote by $R(m, D)$ the expected number of distinct R obtainable. According to Condition (10) of [60], we have $R(m, D) \approx q^{k(2m+1-D)-3}$. Then the condition

$$R(m, D) \gg \frac{q^{k(3m-D)}}{N_{q^k}(m, 3m-D)} \tag{4.18}$$

can ensure that there exists a solution (k_1, k_2, R) for which R is m-smooth.

It is difficult to determine the exact cost $G_{q^k}(m, D)$ of the Gröbner basis finding step. After the Gröbner basis is found, the cost to find an m-smooth R is $(q^k)^{3m-D}/N_{q^k}(m, 3m-D) \cdot S_{q^k}(m, 3m-D)$.

4.3 Computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 1429}}$

We present a concrete analysis of the DLP algorithm described in §4.2 for computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 1429}}$. In fact, this field is embedded in its quadratic extension $\mathbb{F}_{3^{12\cdot 1429}}$, and it is in the latter field where the DLP algorithm of §4.2 is executed. Thus, we have $q = 3^6 = 729$ and $n = 1429$.

As mentioned in §3.1, our main motivation for finding discrete logarithms in $\mathbb{F}_{3^{6\cdot 1429}}$ is to attack the elliptic curve discrete logarithm problem in $E_1(\mathbb{F}_{3^{1429}})$, where E_1 is the supersingular elliptic curve $y^2 = x^3 - x - 1$ with $|E_1(\mathbb{F}_{3^{1429}})| = cr$; here $c = 7622150170693$ is a 43-bit cofactor and $r = (3^{1429} - 3^{715} + 1)/c$ is a 2223-bit prime. The elliptic curve discrete logarithm problem in the order-r subgroup of $E_1(\mathbb{F}_{3^{1429}})$ can be efficiently reduced to the discrete logarithm problem in the order-r subgroup of $\mathbb{F}^*_{3^{12\cdot 1429}}$. In the latter problem, we are given two elements α, β of order r in $\mathbb{F}^*_{3^{12\cdot 1429}}$ and we wish to find $\log_\alpha \beta$. It can be readily seen that $\log_\alpha \beta = (\log_g \beta)/(\log_g \alpha) \bmod r$, where g is a generator of $\mathbb{F}^*_{3^{12\cdot 1429}}$. Thus, we will henceforth assume that h has order r and that we only need to find $\log_g h \bmod r$. An immediate consequence of this restriction is that all the linear algebra in the useful algorithm can be performed modulo the 2223-bit r instead of modulo the 27179-bit number $N = 3^{12\cdot 1429} - 1$.

The parameters for each step of the algorithm were carefully chosen in order to balance the running time of the steps. We also took into account the degree to which each step could be parallelized on conventional computers. A summary of the parameter choices for the descent is given in Figure 4.1. The cost of each step is given in Table 4.2.

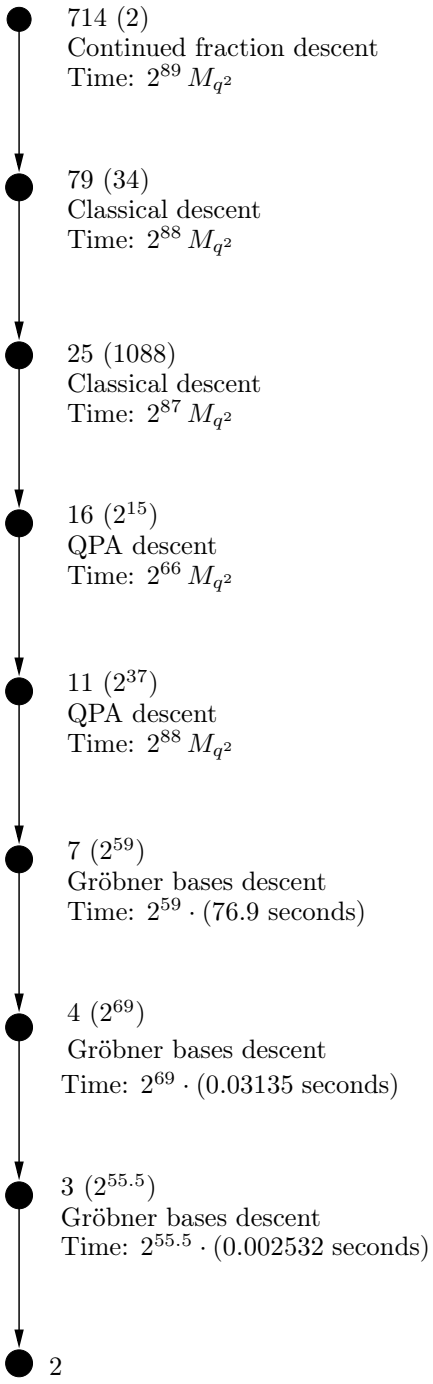

Figure 4.1: A typical path of the descent tree for computing an individual logarithm in $\mathbb{F}_{3^{12\cdot 1429}}$ ($q = 3^6$). The numbers in parentheses next to each node are the expected number of nodes at that level. 'Time' is the expected time to generate all nodes at a level.

Finding logarithms of linear polynomials		
Relation generation	$2^{30}M_{q^2}$	$2^{30}M_{q^2}$
Linear algebra	$2^{48}A_r$	$2^{51}M_{q^2}$
Finding logarithms of irreducible quadratic polynomials		
Relation generation	$3^{12}\cdot 2^{39}M_{q^2}$	$2^{58}M_{q^2}$
Linear algebra	$3^{12}\cdot 2^{48}A_r$	$2^{70}M_{q^2}$
Descent		
Continued-fractions (714 to 79)	$2^{89}M_{q^2}$	$2^{89}M_{q^2}$
Classical (79 to 25)	$34\cdot 2^{83}M_{q^2}$	$2^{88}M_{q^2}$
Classical (25 to 16)	$1088\cdot 2^{77}M_{q^2}$	$2^{87}M_{q^2}$
QPA (16 to 11)	$2^{15}\cdot(2^{46}M_{q^2}+2^{48}A_r)$	$2^{66}M_{q^2}$
QPA (11 to 7)	$2^{37}\cdot(2^{45}M_{q^2}+2^{48}A_r)$	$2^{88}M_{q^2}$
Gröbner bases (7 to 4)	$2^{59}\cdot(76.9$ seconds$)$	$2^{95.3}M_{q^2}$
Gröbner bases (4 to 3)	$2^{69}\cdot(0.03135$ seconds$)$	$2^{94}M_{q^2}$
Gröbner bases (3 to 2)	$2^{55.5}\cdot(0.002532$ seconds$)$	$2^{77}M_{q^2}$

Table 4.2: Estimated costs of the main steps of the useful DLP algorithm for computing discrete logarithms in $\mathbb{F}_{3^{12\cdot 1429}}$ ($q=3^6$). A_r and M_{q^2} denote the costs of an addition modulo the 2223-bit prime r and a multiplication in $\mathbb{F}_{3^{12}}$. We use the cost ratio $A_r/M_{q^2}=2^3$, and also assume that 2^{30} multiplications in $\mathbb{F}_{3^{12}}$ can be performed in 1 second (see §4.3.8).

4.3.1 Setup

We chose the representations $\mathbb{F}_{3^6}=\mathbb{F}_3[U]/(U^6+2U^4+U^2+2U+2)$ and $\mathbb{F}_{3^{12}}=\mathbb{F}_{3^6}[V]/(V^2+U^{365})$. We selected $h_0=(U^{265}V+U^{236})X^2+(U^{160}V+U^{24})X+(U^{628}V+U^{293})\in\mathbb{F}_{3^{12}}[X]$ and $h_1=1$, and $I_X\in\mathbb{F}_{3^{12}}[X]$ to be the degree-1429 monic irreducible factor of $h_1(X^{3^6})\cdot X-h_0(X^{3^6})$. The other irreducible factors have degrees 5, 5 and 19.

4.3.2 Finding logarithms of linear polynomials

The factor base $\mathcal{B}_1$ has size $3^{12}\approx 2^{19}$. The cost of relation generation is approximately $2^{30}M_{q^2}$, whereas the cost of the linear algebra is approximately $2^{48}A_r$.

4.3.3 Finding logarithms of irreducible quadratic polynomials

For each $u\in\mathbb{F}_{3^{12}}$, the expected cost of computing logarithms of all quadratics in $\mathcal{B}_{2,u}$ is $2^{39}M_{q^2}$ for the computation of $\mathcal{H}(Q)$, and $2^{48}A_r$ for the linear algebra.

4.3.4 Continued-fractions descent

For the continued-fractions descent, we selected $m=79$. The expected cost of this descent is $2^{89}M_{q^2}$. The expected number of distinct irreducible factors of degree (at most) 79 obtained is $2A_{3^{12}}(79,714)\approx 34$.

4.3.5 Classical descent

Two classical descent stages are employed. In the first stage, which uses the alternative method described in §3.3.5, we have $D = 79$ and select $m = 25$, $s = 5$, $\delta = 2$, which yield $t_1 = 247$ and $t_2 = 284$. The expected cost of the descent for each of the 34 degree-79 polynomials is approximately $2^{58.7} \cdot S_{q^2}(25, 205)$. The expected total number of distinct irreducible polynomials of degree (at most) 25 obtained is approximately 1088.

In the second classical descent stage, which uses the first method described in §3.3.5, we have $D = 25$ and select $m = 16$, $s = 2$, $\delta = 2$, which yield $t_1 = 176$ and $t_2 = 127$. The expected cost of the descent for each of the 1088 degree-25 polynomials is approximately $2^{54.4} \cdot S_{q^2}(16, 127)$. The expected total number of distinct irreducible polynomials of degree (at most) 16 obtained is approximately 2^{15}.

4.3.6 QPA descent

Two QPA descent stages are employed. In the first stage, we have $D = 16$ and select $m = 11$. For each Q, the expected cost of relation generation is $2^{30.9} \cdot S_{q^2}(11, 48)$ and the cost of the linear algebra is $2^{48} A_r$. Also for each Q, the expected number of distinct polynomials of degree at most 11 obtained is expected to be $A_{q^2}(11, 48) \cdot q^2 \approx 2^{22}$. Thus, the total number of distinct polynomials of degree at most 11 obtained after the first QPA descent stage is approximately 2^{37}.

In the second stage, we have $D = 11$ and select $m = 7$. For each Q, the expected cost of relation generation is $2^{28.4} \cdot S_{q^2}(7, 33)$ and the cost of the linear algebra is $2^{48} A_r$. Also for each Q, the expected number of distinct polynomials of degree at most 7 obtained is expected to be $A_{q^2}(7, 33) \cdot q^2 \approx 2^{22}$. Thus, the total number of distinct polynomials of degree at most 7 obtained after the second QPA descent stage is approximately 2^{59}.

4.3.7 Gröbner bases descent

Three Gröbner bases descent stages are employed. The first stage has $D = 7$ and $m = 4$, and is expected to yield approximately 2^{69} polynomials of degree (at most) 4. The second stage has $D = 4$ and $m = 3$, and is expected to yield all the $2^{55.5}$ monic irreducible polynomials of degree 3. The third stage has $D = 3$ and $m = 2$ and is applied to the $2^{55.5}$ monic irreducible cubics.

For all three stages, we use the experimental results from §3.3.7. The experiments were run on a 2.9 GHz Intel core i7-3520M using Magma's implementation of Faugère's F4 algorithm [47].

4.3.8 Overall running time

The second column of Table 4.2 gives the running time estimates for the main steps of the useful DLP algorithm in three units of time: A_r, M_{q^2}, and seconds. In order to assess the overall time, we make some assumptions about the ratios of these units of time.

First, we shall assume that $A_r/M_{q^2} = 2^3$. To justify this, we observe that a 2223-bit integer can be stored in 35 64-bit words. The X86-64 instruction set has an **ADD** operation that adds two 64-bit unsigned integers in one clock cycle. Hence, integer addition can be completed in 35 clock cycles. Modular reduction comprises one conditional statement plus

one subtraction (required in roughly half of all modular additions). One can use a lazy reduction technique that amortizes the cost of a modular reduction among many integer additions. All in all, the cost of A_r can be estimated to be 35 clock cycles. Unlike for 64-bit integer multiplication, there is no native support for $\mathbb{F}_{3^{12}}$ multiplication on an Intel Core i7 machine. However, we expect that a specially designed multiplier could be built to achieve a multiplication cost of 4 clock cycles. This gives us an A_r/M_{q^2} ratio of approximately 2^3.

Next, since a multiplication in M_{q^2} can be done in 4 clock cycles, we will transform one second on a 2.9 GHz machine (on which the Gröbner bases descent experiments were performed) into $2^{30}M_{q^2}$.

Using these estimates, we see from the third column of Table 4.2 that the overall running time of the useful algorithm is approximately $2^{95.8}M_{q^2}$. We note that the relation generation, continued-fractions descent, classical descent, and Gröbner bases descent steps, and also the relation generation portion of QPA descent, are effectively parallelizable in the sense that one can essentially achieve a factor-C speedup if C processors are available. Moreover, the linear system of equations for finding logarithms of linear polynomials, the $3^{12} \approx 2^{19}$ linear systems of equations for finding logarithms of irreducible quadratic polynomials, and the $2^{15} + 2^{37}$ linear systems of equations in QPA descent are also effectively parallelizable since each linear system can be solved in less than one day using a small number of GPUs and CPUs (see [74] and [14]).

4.3.9 Comparisons

The upper bound of $2^{95.8}M_{q^2}$ on the running time of the useful algorithm for computing logarithms in $\mathbb{F}_{2^{6\cdot 1429}}$ convincingly demonstrates that this field offers drastically less security than the 2^{192} resistance to attacks by Coppersmith's algorithm [40, 94]. The decrease in security is even more pronounced when one considers that Coppersmith's algorithm is not parallelizable since a dominant step is the solution of an enormous system of linear equations, whereas the useful algorithm is effectively parallelizable.

In §5.4, we outline how the estimates of $2^{95.8}M_{q^2}$ for computing logarithms in $\mathbb{F}_{3^{6\cdot 1429}}$ can be reduced to $2^{78.8}M_{q^2}$. The improved estimates use techniques from [59], [60], that were developed after the estimates in this section were derived.

4.4 Solving the discrete logarithm problem in $\mathbb{F}_{3^{6\cdot 137}}$

The supersingular elliptic curve $E : y^2 = x^3 - x + 1$ has order

$$|E(\mathbb{F}_{3^{137}})| = cr,$$

where

$$c = 7 \cdot 4111 \cdot 5729341 \cdot 42526171$$

and

$$r = (3^{137} - 3^{69} + 1)/c = 330982801190901910287755800550821750564284956 23$$

is a 155-bit prime. The Weil and Tate pairing attacks [100, 49] efficiently reduce the discrete logarithm problem in the order-r subgroup $\mathcal{E}$ of $E(\mathbb{F}_{3^{137}})$ to the discrete logarithm problem in the order-r subgroup $\mathbb{G}$ of $\mathbb{F}^*_{3^{6\cdot 137}}$.

Our approach to computing discrete logarithms in $\mathbb{G}$ is to use Joux's algorithm which is the algorithm described in §4.2 but without any QPA descent steps (§4.2.6). We compute logarithms in the quadratic extension $\mathbb{F}_{3^{12\cdot 137}}$ of $\mathbb{F}_{3^{6\cdot 137}}$ (so $q = 3^4$, $n = 137$ and $k = 3$ in the notation of §4.2). With this setting, we do not need to compute the logarithms of the quadratic polynomials as described in §4.2.3, since these can be computed on the fly using the 2-to-1 descent method outlined in §4.4.2.

Consider we are given two elements α, β of order r in $\mathbb{F}^*_{3^{12\cdot 137}}$ and we wish to find $\log_\alpha \beta$. Let g be a generator of $\mathbb{F}^*_{3^{12\cdot 137}}$. Then $\log_\alpha \beta = (\log_g \beta)/(\log_g \alpha) \bmod r$. Thus, in the remainder of the section we will assume that we need to compute $\log_g h \bmod r$, where h is an element of order r in $\mathbb{F}^*_{3^{12\cdot 137}}$.

The DLP instance we solved is described in §4.4.1. In §4.4.2, the 2-to-1 descent method is presented. The concrete estimates from §4.2 for solving the DLP instance are given in §4.4.4. These estimates are only upper bounds on the running time of the algorithm. Nevertheless, they provide convincing evidence for the feasibility of the discrete logarithm computations. Our experimental results are presented in §4.4.5.

4.4.1 Problem instance

Let N denote the order of $\mathbb{F}^*_{3^{12\cdot 137}}$. Using the tables from the Cunningham Project [42], we determined that the factorization of N is $N = p_1^4 \cdot \prod_{i=2}^{31} p_i$, where the p_i are the following primes (and $r = p_{25}$):

$$
\begin{aligned}
p_1 &= 2 \quad p_2 = 5 \quad p_3 = 7 \quad p_4 = 13 \quad p_5 = 73 \quad p_6 = 823 \quad p_7 = 4111 \quad p_8 = 4933\\
p_9 &= 236737 \quad p_{10} = 344693 \quad p_{11} = 2115829 \quad p_{12} = 5729341 \quad p_{13} = 42526171\\
p_{14} &= 217629707 \quad p_{15} = 634432753 \quad p_{16} = 685934341 \quad p_{17} = 82093596209179\\
p_{18} &= 4354414202063707 \quad p_{19} = 18329390240606021 \quad p_{20} = 46249052722878623693\\
p_{21} &= 201820452878622271249 \quad p_{22} = 11393882913488022495414289252647 7\\
p_{23} &= 518545466463281867910174177004304863965 13\\
p_{24} &= 27353706568336941255688896404282780237 6371\\
p_{25} &= 33098280119090191028775580055082175056428495623\\
p_{26} &= 70671225820194025466782664267300876838722911504837 9\\
p_{27} &= 10808180977383999518825680049914154368439303545035055 1\\
p_{28} &= 91321974595662761339222271626247966116126450162880692588587183952237\\
p_{29} &= 39487531149773489532096996293368370182957526257988573877031054477249\\
&\quad 393549\\
p_{30} &= 40189860022384850044254854796561182547553072730738823866986300807613\\
&\quad 29207749418522920289\\
p_{31} &= 19064323153825272072803685870803955622834286523139037403580752310822\\
&\quad 78966446469840637369426240662274068981321133662265931584644197 13.
\end{aligned}
$$

We chose $\mathbb{F}_{3^4} = \mathbb{F}_3[U]/(U^4 + U^2 + 2)$ and $\mathbb{F}_{3^{12}} = \mathbb{F}_{3^4}[V]/(V^3 + V + U^2 + U)$, and selected $h_0(X) = V^{326196}X^2 + V^{35305}X + V^{204091} \in \mathbb{F}_{3^{12}}[X]$ and $h_1 = 1$. Then $I_X \in \mathbb{F}_{3^{12}}[X]$ is the

degree-137 monic irreducible factor of $X - h_0(X^{3^4})$; the other irreducible factor has degree 25.

We chose the generator $g = X + V^{113713}$ of $\mathbb{F}^*_{3^{12\cdot 137}}$. To generate an order-$r$ discrete logarithm challenge h, we computed

$$h' = \sum_{i=0}^{136} \left(V^{\lfloor \pi \cdot (3^{12})^{i+1} \rfloor \bmod 3^{12}} \right) X^i$$

and then set $h = (h')^{N/r}$. The discrete logarithm $\log_g h \bmod r$ was found to be

$$x = 273396190769750939202455159732141869630256565 59.$$

This can be verified by checking that $h = (g^{N/r})^y$, where $y = x \cdot (N/r)^{-1} \bmod r$.

4.4.2 2-to-1 descent

The Gröbner bases descent methodology of §4.2.7 can be employed in the case $(D, m) = (2, 1)$. However, as also reported by Joux in his $\mathbb{F}_{2^{6168}}$ discrete log computation [78], we found the descent to be successful for only about 50% of all irreducible quadratic polynomials. In fact, the expected number of distinct obtainable R (defined in §4.2.7) for a given quadratic polynomial is exactly one. But recall that for each quadratic polynomial one needs to solve a *zero-dimensional* quadratic multivariate system. We observed that this system is always in *shape position*[2] and the univariate polynomial of its Gröbner basis is of degree 2. Then, whenever the system is consistent, two distinct polynomials R will be found for a single quadratic polynomial. Thus, only about half of quadratic polynomials take the totality of the available polynomials R. However, despite this fact, some strategies can be used to increase the percentage of successful descents.

Let $Q(X) \in \mathbb{F}_{q^3}[X]$ be an irreducible quadratic polynomial for which the Gröbner bases descent method failed.

Strategy 4.1. Introduced by Joux [78] and Göloğlu et al. [58], this strategy is based on the systematic equation derived from $Y^{q'} - Y$ where $q' < q$ and $\mathbb{F}_{q'}$ is a proper subfield of $\mathbb{F}_{q^3}$ instead of the systematic equation (3.2) derived from $Y^q - Y$. Let p be the characteristic of $\mathbb{F}_q$, and let $q = p^\ell$, $q' = p^{\ell'}$, and $s = \ell - \ell'$. Then $q = p^s \cdot q'$. Now, using standard Gröbner bases techniques, one searches for $(a, b, c, d) \in \mathcal{P}_{q',4}$ such that

$$G = (aX + b)(\overline{c}\overline{h}_0 + \overline{d}\,\overline{h}_1)^{p^s} - (\overline{a}\overline{h}_0 + \overline{b}\,\overline{h}_1)^{p^s}(cX + d) = QR$$

with $R \in \mathbb{F}_{q^3}[X]$. Note that $\deg R \leq 2p^s - 1$. For our discrete logarithm computations in $\mathbb{F}_{3^{6\cdot 137}}$ and $\mathbb{F}_{3^{6\cdot 163}}$ (§4.5), we have $q = 3^4$ and used $q' = 3^3$, so $s = 1$ and $\deg R = 5$. If R is

[2]Let $S = \{f_1, \ldots, f_s\} \in \mathbb{K}[X_1, \ldots, X_\ell]^s$ be a *zero-dimensional* multivariate system over a field $\mathbb{K}$. The system S is said to be in *shape position* if its *reduced* Gröbner basis, with respect to the *lexicographic* order, is of the form $[X_1 - g_1(X_\ell), \ldots, X_{\ell-1} - g_{\ell-1}(X_\ell), g_\ell(X_\ell)]$, where $g_1, \ldots, g_\ell$ are univariate polynomials in $\mathbb{K}[X]$. g_ℓ is called the univariate polynomial of the Gröbner basis.

1-smooth, then we obtain a linear relationship between $\log_g Q$ and logs of linear polynomials since

$$G^q \equiv \overline{h}_1^{p^s q} \cdot (cX+d)^{p^s} \cdot \prod_{\alpha \in \mathbb{F}_{q'}} \left((aX+b)^{p^s} - \alpha(cX+d)^{p^s}\right) \pmod{I_X},$$

as can be seen by making the substitution $Y \mapsto (aX+b)^{p^s}/(cX+d)^{p^s}$ into the systematic equation derived from $Y^{q'} - Y$.

Unfortunately, in all instances we considered the polynomial R never factors completely into linear polynomials (see §4.4.3 for a detailed explanation). However, it hopefully factors into a quadratic polynomial Q' and $2p^s - 3$ linear polynomials, thereby yielding a relation between Q and another quadratic which has a roughly 50% chance of descending using Gröbner bases descent. Combined with the latter, we observed that this strategy descends about 95% of all irreducible quadratic polynomials in the fields $\mathbb{F}_{3^{6\cdot 137}}$ and $\mathbb{F}_{3^{6\cdot 163}}$.

Strategy 4.2. Let $Q(X)$ be written as $Q(X) = X^2 + uX + v$. Then we have

$$\begin{aligned} \overline{h}_1^{2q} Q(X) \equiv\ & \overline{h}_1^{2q} Q((\overline{h}_0/\overline{h}_1)^q) \ =\ \overline{h}_0^{2q} + u\overline{h}_0^q \overline{h}_1^q + v\overline{h}_1^{2q} \\ =\ & (\overline{h}_0^2 + \overline{u}\overline{h}_0\overline{h}_1 + \overline{v}\,\overline{h}_1^2)^q \pmod{I_X}. \end{aligned} \tag{4.19}$$

It can be seen that the degree-4 polynomial $f_Q(X) = \overline{h}_0^2 + \overline{u}\overline{h}_0\overline{h}_1 + \overline{v}\,\overline{h}_1^2$ is either a product of two irreducible quadratics or itself irreducible.[3] In the former case, which happens with probability about one half (see [82, Lemma 2]), we apply the standard Gröbner bases descent method to the two irreducible quadratics. If both descents are successful, then we have succeeded in descending the original Q.

The strategies are combined in the following manner. For an irreducible quadratic $Q \in \mathbb{F}_{q^3}[X]$, we first check if the Gröbner bases descent is successful. If the descent fails, we apply Strategy 4.2 to Q. In the case where f_Q factors into two irreducible quadratics, and at least one of them fails to descent with Gröbner bases descent, we apply Strategy 4.1 to Q. If Strategy 4.1 fails on Q, we apply it to the two quadratic factors of f_Q. In the case where f_Q is irreducible, we apply Strategy 4.1 to Q.

If none of the attempts succeed, we declare Q to be "bad", and avoid it in the higher-degree descent steps by repeating a step until all the quadratics encountered are "good". In our experiments with $\mathbb{F}_{3^{6\cdot 137}}$ and $\mathbb{F}_{3^{6\cdot 163}}$, we observed that approximately 97.2% of all irreducible quadratic polynomials Q were "good".

To see that this percentage is sufficient to complete the descent phase in these two fields, consider a 3-to-2 descent step where the number of resulting irreducible quadratic polynomials

[3] More generally, for polynomials F, g_0, g_1 and P over a field $\mathbb{F}$, if both F and P are irreducible over $\mathbb{F}$ and $P \mid g_1^{\deg F} F(g_0/g_1)$, then we have $\deg F \mid \deg P$. Indeed, one can easily check that the map $\phi : \mathbb{K} = \mathbb{F}/(F) \to \mathbb{L} = \mathbb{F}/(P)$ with $\phi(H + (F)) = g_1^{\deg H} H(g_0/g_1) + (P)$ for all polynomials H over $\mathbb{F}$ is a well-defined field homomorphism and thereby one-to-one. Thus, one can assume that $\mathbb{F} \subset \mathbb{K} \subset \mathbb{L}$ and get $\deg P = [\mathbb{L} : \mathbb{F}] = [\mathbb{L} : \mathbb{K}] \cdot [\mathbb{K} : \mathbb{F}] = [\mathbb{L} : \mathbb{K}] \cdot \deg F$.

is 42 on average (see equation (4.16)). Then the probability of descending a degree-3 polynomial after finding one useful solution (k_1, k_2, R) in Gröbner bases descent is $0.972^{42} \approx 0.3$. Therefore, after at most four trials we expect to successfully descend a degree-3 polynomial. Since the expected number of distinct solutions of (4.17) is approximately q^3 (according to Condition (10) of [60]), one can afford this many trials.

4.4.3 A remark on Strategy 4.1

In this section, the notations of Strategy 4.1 are used. Let $(a, b, c, d) \in \mathcal{P}_{q',4}$, and let

$$G = (aX + b)(\overline{c}\overline{h}_0 + \overline{d}\,\overline{h}_1)^{p^s} - (\overline{a}\overline{h}_0 + \overline{b}\,\overline{h}_1)^{p^s}(cX + d).$$

The derivative of G is given as

$$G' = \left((\widetilde{a}\overline{c} - \widetilde{c}\overline{a})\overline{h}_0 + (\widetilde{a}\overline{d} - \widetilde{c}\overline{b})\overline{h}_1\right)^{p^s}, \tag{4.20}$$

where for $\gamma \in \mathbb{F}_{q^3}$, $\widetilde{\gamma}$ denotes the element γ^{q^3/p^s}. Thus, we obtain

$$G = XG' + \left((\widetilde{b}\overline{c} - \widetilde{d}\overline{a})\overline{h}_0 + (\widetilde{b}\overline{d} - \widetilde{d}\overline{b})\overline{h}_1\right)^{p^s}. \tag{4.21}$$

Now, suppose that G has a repeated root λ in some extension field of $\mathbb{F}_{q^3}$. Then λ is also a root of G' and consequently a root of both

$$(\widetilde{a}\overline{c} - \widetilde{c}\overline{a})\overline{h}_0 + (\widetilde{a}\overline{d} - \widetilde{c}\overline{b})\overline{h}_1 \quad \text{and} \quad (\widetilde{b}\overline{c} - \widetilde{d}\overline{a})\overline{h}_0 + (\widetilde{b}\overline{d} - \widetilde{d}\overline{b})\overline{h}_1.$$

This gives

$$\frac{\overline{h}_0(\lambda)}{\overline{h}_1(\lambda)} = \frac{\widetilde{a}\overline{c} - \widetilde{c}\overline{a}}{\widetilde{a}\overline{d} - \widetilde{c}\overline{b}} = \frac{\widetilde{b}\overline{c} - \widetilde{d}\overline{a}}{\widetilde{b}\overline{d} - \widetilde{d}\overline{b}}.$$

Therefore, the equation

$$(\widetilde{a}\overline{c} - \widetilde{c}\overline{a})(\widetilde{b}\overline{d} - \widetilde{d}\overline{b}) = (\widetilde{b}\overline{c} - \widetilde{d}\overline{a})(\widetilde{a}\overline{d} - \widetilde{c}\overline{b})$$

yields

$$(\widetilde{a}\widetilde{d} - \widetilde{b}\widetilde{c})(\overline{a}\overline{d} - \overline{b}\overline{c}) = 0,$$

which means that $ad - bc = 0$. This contradicts the fact that (a, b, c, d) is in $\mathcal{P}_{q',4}$. Hence G has no repeated root.

Let's assume that the degree of G is $2p^s + 1$, implying that G' has degree $2p^s$. We recall the following result of Swan [118] and apply it to G.

Theorem 4.3 (Corollary 1, [118])**.** Let $\mathbb{K}$ be an odd-characteristic finite field. Let $F \in \mathbb{K}[X]$ be a polynomial with no repeated root. Then the degree of F and the number of irreducible factors of F have same parity if and only if the discriminant of F is a square in $\mathbb{K}$.

Assume that p is odd. To compute the discriminant $D(G)$ of G, we use Swan's formula

$$D(G) = (-1)^{\deg G\left(\frac{\deg G-1}{2}+\deg G'\right)}\mathrm{lc}(G^{-1})\mathrm{lc}(G')^{\deg G}\prod_{i=1}^{\deg G'} G(x_i), \tag{4.22}$$

where $x_1, \ldots, x_{\deg G'}$ are the roots of G' (counted with multiplicity), and $\mathrm{lc}(G)$ denotes the leading coefficient of G. Since $\deg G' = 2p^s$, we have from (4.20) that G' has two roots λ_1 and λ_2, each with multiplicity p^s. Then (4.22) becomes

$$D(G) = \mathrm{lc}(G)^{2p^s}(G(\lambda_1)G(\lambda_2))^{p^s}.$$

Thus, $D(G)$ is a square in $\mathbb{F}_{q^3}$ if and only if $G(\lambda_1)G(\lambda_2)$ is a square in $\mathbb{F}_{q^3}$, and at the same time, by (4.21), if and only if γ is a square in $\mathbb{F}_{q^3}$, where γ is given as

$$\gamma = \left((\widetilde{b}\overline{c} - \widetilde{d}\overline{a})\overline{h}_0(\lambda_1) + (\widetilde{b}\overline{d} - \widetilde{d}\overline{b})\overline{h}_1(\lambda_1)\right)\left((\widetilde{b}\overline{c} - \widetilde{d}\overline{a})\overline{h}_0(\lambda_2) + (\widetilde{b}\overline{d} - \widetilde{d}\overline{b})\overline{h}_1(\lambda_2)\right). \tag{4.23}$$

In (4.20), notice that $\widetilde{a}\overline{c} - \widetilde{c}\overline{a}$ and $\widetilde{a}\overline{d} - \widetilde{c}\overline{b}$ cannot be simultaneously zero since $G' \neq 0$. Let's assume that $\widetilde{a}\overline{c} - \widetilde{c}\overline{a} \neq 0$, and write

$$\beta_0 = \widetilde{a}\overline{c} - \widetilde{c}\overline{a}, \quad \beta_1 = \widetilde{a}\overline{d} - \widetilde{c}\overline{b} \quad \text{and} \quad \delta = \beta_1/\beta_0.$$

Since λ_1 and λ_2 are the roots of $\beta_0\overline{h}_0 + \beta_1\overline{h}_1$, we have for $i = 1, 2$

$$\overline{h}_0(\lambda_i) = \delta\overline{h}_1(\lambda_i).$$

After replacing $\overline{h}_0(\lambda_i)$ by $\delta\overline{h}_1(\lambda_i)$ in (4.23), one sees that γ is a square in $\mathbb{F}_{q^3}$ if and only if $\overline{h}_1(\lambda_1)\overline{h}_1(\lambda_2)$ is a square in $\mathbb{F}_{q^3}$. This shows that with our choice $h_1 = 1$, G never factors into an even number of irreducible polynomials, and thereby R never factors into linear polynomials. In §4.5, the choice $h_0 = 1$ leads to the same conclusion.

Now, let $\overline{h}_0$, $\overline{h}_1$ be written as $\overline{h}_0 = r_2X^2 + r_1X + r_0$, $\overline{h}_1 = s_2X^2 + s_1X + s_0$. Since λ_1 and λ_2 are the roots of

$$\beta_0\overline{h}_0 + \beta_1\overline{h}_1 = \beta_0\left((r_2 + \delta s_2)X^2 + (r_1 + \delta s_1)X + r_0 + \delta s_0\right),$$

we have

$$\lambda_1\lambda_2 = \frac{r_0 + \delta s_0}{r_2 + \delta s_2} \quad \text{and} \quad \lambda_1 + \lambda_2 = -\frac{r_1 + \delta s_1}{r_2 + \delta s_2}. \tag{4.24}$$

(Note that our assumption $\deg G' = 2p^s$ ensures that $r_2 + \delta s_2 \neq 0$.) Hence, expanding $\overline{h}_1(\lambda_1)\overline{h}_1(\lambda_2)$ and substituting $\lambda_1\lambda_2$, $\lambda_1 + \lambda_2$ by their values in (4.24), we obtain

$$\overline{h}_1(\lambda_1)\overline{h}_1(\lambda_2) = \frac{1}{(r_2 + \delta s_2)^2}\left[(r_0s_2 - r_2s_0)^2 - r_0r_1s_1s_2 + r_1^2s_0s_2 - r_1r_2s_0s_1 + r_0r_2s_2^2\right]. \tag{4.25}$$

Consequently, to expect having R splitting into linear polynomials, $\overline{h}_0 = r_2X^2 + r_1X + r_0$, $\overline{h}_1 = s_2X^2 + s_1X + s_0$ must be chosen such that $(r_0s_2 - r_2s_0)^2 - r_0r_1s_1s_2 + r_1^2s_0s_2 - r_1r_2s_0s_1 + r_0r_2s_2$ is a square in $\mathbb{F}_{q^3}$.

4.4.4 Estimates

The factor base $\mathcal{B}_1$ has size $3^{12} \approx 2^{19}$. The cost of the relation generation is approximately $2^{29.2} M_{q^3}$, whereas the cost of the linear algebra is approximately $2^{44.4} A_r$. Figure 4.2 shows the estimated running times for the descent stage. Further information about the parameter choices are provided below.

1. For the continued-fractions descent stage, we selected $m = 13$. The expected cost of this descent is $2^{43.2} M_{q^3}$, and the expected number of irreducible factors of degree (at most) 13 obtained is $2A_{3^{12}}(68, 13) \approx 20$.

2. Two classical descent stages are employed. In the first stage, we have $D = 13$ and select $m = 7$, $s = 3$, $\delta = 1$, which yield $t_1 = 43$ and $t_2 = 34$. The expected cost of the descent for each of the 20 degree-13 polynomials is approximately $2^{33.7} M_{q^3}$. The expected total number of distinct irreducible polynomials of degree (at most) 7 obtained is approximately 320.

   In the second classical descent stage, we have $D = 7$ and select $m = 5$, $s = 3$, $\delta = 1$, which yield $t_1 = 25$ and $t_2 = 31$. The expected cost of the descent for each of the 320 degree-7 polynomials is approximately $2^{34.8} M_{q^3}$. The expected total number of distinct irreducible polynomials of degree (at most) 5 obtained is approximately $5,120$.

3. Our implementation of the Gröbner bases descent stage used Magma's implementation of Faugére's F4 algorithm [47] and took 26.5 minutes on average for a 5-to-3 descent, 34.7 seconds for a 3-to-2 descent, and 0.216 seconds for a 2-to-1 descent. The total expected running time for each of these stages is 94, 211 and 168 days, respectively.

Since all the descent stages can be effectively parallelized, our estimates suggest that a discrete logarithm can be computed in a week or so given a few dozen processors. In fact (and as confirmed by our experimental results), the actual running time is expected to be significantly less than the estimated running time since the estimates are quite conservative; for example, our estimates for the number of branches in a descent step assumes that each distinct irreducible polynomial has degree exactly m, whereas in practice many of these polynomials will have degree significantly less than m.

4.4.5 Experimental results

Our experiments were run on an Intel i7-2600K 3.40 GHz machine (Sandy Bridge), and on an Intel i7-4700MQ 2.40 GHz machine (Haswell).

Relation generation took 1.05 CPU hours (Sandy Bridge, 1 core). The resulting sparse linear system of linear equation was solved using Magma's multi-threaded parallel version of the Lanczos algorithm; the computation took 556.8 CPU hours (Sandy Bridge, 4 cores).

In the continued-fractions descent stage, the first degree-68 polynomial yielded 9 irreducible factors of degrees 12, 12, 11, 10, 8, 6, 6, 2, 1, and the second degree-68 polynomial yielded 11 irreducible factors of degrees 13, 12, 10, 10, 7, 6, 5, 2, 1, 1, 1. The computation took 22 CPU hours (Haswell, 4 cores).

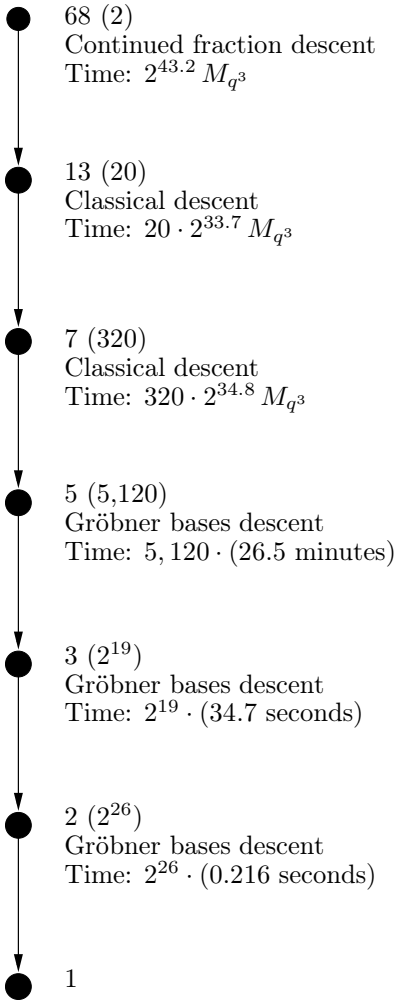

Figure 4.2: A typical path of the descent tree for computing an individual logarithm in $\mathbb{F}_{3^{12\cdot 137}}$ ($q = 3^4$). The numbers in parentheses next to each node are the expected number of nodes at that level. 'Time' is the expected time to generate all nodes at a level.

Classical descent was used on the 9 polynomials of degree ≥ 8 to obtain polynomials of degree ≤ 7, and then on the 23 polynomials of degree 7 and 23 polynomials of degree 6 to obtain polynomials of degree ≤ 5. These computations took 80 CPU hours (Haswell, 4 cores).

Finally, we used 5-to-3, 4-to-3, 3-to-2 and 2-to-1 Gröbner bases descent procedures. The average time for a 4-to-3 descent was 33.8 seconds; the other average times are given in Figure 4.2. In total, we performed 233 5-to-3 descents, 174 4-to-3 descents, and 11573 3-to-2 descents. These computations took 115.2 CPU hours, 1.5 CPU hours, and 111.2 CPU hours, respectively (Haswell, 4 cores). We also performed 493537 2-to-1 descents; their running times are incorporated into the running times for the higher-level descents.

4.5 Solving the discrete logarithm problem in $\mathbb{F}_{3^{6\cdot 163}}$

The supersingular elliptic curve $E : y^2 = x^3 - x - 1$ has order

$$|E(\mathbb{F}_{3^{163}})| = 3^{163} + 3^{82} + 1 = r$$

where r is the following 259-bit prime:

$$r = 589881151426658740854227725580736348850640632297373414091790995505756623268837.$$

The Weil and Tate pairing attacks [100, 49] efficiently reduce the discrete logarithm problem in the order-r group $\mathcal{E} = E(\mathbb{F}_{3^{163}})$ to the discrete logarithm problem in the order-r subgroup $\mathbb{G}$ of $\mathbb{F}^*_{3^{6\cdot 163}}$.

As in §4.4, we will compute logarithms in $\mathbb{G}$ by using Joux's algorithm to compute logarithms in the quadratic extension $\mathbb{F}_{3^{12\cdot 163}}$ of $\mathbb{F}_{3^{6\cdot 163}}$ (so $q = 3^4$ and $n = 163$ in the notation of §4.2). We will compute $\log_g h \bmod r$, where g is a generator of $\mathbb{F}^*_{3^{12\cdot 163}}$ and h is an element of order r in $\mathbb{F}^*_{3^{12\cdot 163}}$.

4.5.1 Problem instance

Let N denote the order of $\mathbb{F}^*_{3^{12\cdot 163}}$. Using the tables from the Cunningham Project [42], we partially factored N as $N = C \cdot p_1^4 \cdot \prod_{i=2}^{22} p_i$, where the p_i are the following primes (and

$r = p_{20}$):

$$p_1 = 2 \quad p_2 = 5 \quad p_3 = 7 \quad p_4 = 13 \quad p_5 = 73 \quad p_6 = 653 \quad p_7 = 50857$$
$$p_8 = 107581 \quad p_9 = 489001 \quad p_{10} = 105451873 \quad p_{11} = 380998157$$
$$p_{12} = 8483499631 \quad p_{13} = 5227348213873 \quad p_{14} = 8882811705390167$$
$$p_{15} = 4956470591980320134353 \quad p_{16} = 23210817035829275705929$$
$$p_{17} = 350717106095718676799491213620033814689659449$$
$$p_{18} = 635188514196405741125949952661184826072045955243$$
$$p_{19} = 842687359180941058363182465115337641211400104811307410674430711031488176 17701717$$
$$p_{20} = 589881151426658740854227725580736348850640632297373414091790995505756623268837$$
$$p_{21} = 132629057840437233700340256676181210815404382831772686800451868848532620412724278105428771691382890569577153531961762590484982180238880127520163287477737925665237296550978481021134887956989367683944949926212312022819011019340957620502000045691081669475648919901346991751981450831153457094555852222882729833782621504374409486151475445415149317 7.$$

We verified that $\gcd(C, N/C) = 1$ and that C is not divisible by any of the first 10^7 primes. Consequently, if an element g is selected uniformly at random from $\mathbb{F}^*_{3^{12\cdot 163}}$, and g satisfies $g^{N/p_i} \neq 1$ for $1 \leq i \leq 22$, then g is a generator with very high probability.[4]

We chose $\mathbb{F}_{3^4} = \mathbb{F}_3[U]/(U^4 + U^2 + 2)$ and $\mathbb{F}_{3^{12}} = \mathbb{F}_{3^4}[V]/(V^3 + V + U^2 + U)$, and selected $h_0(X) = 1$ and

$$h_1(X) = X^2 + V^{530855} \in \mathbb{F}_{3^{12}}[X].$$

Then $I_X \in \mathbb{F}_{3^{12}}[X]$ is the degree-163 irreducible polynomial $X \cdot h_1(X^{3^4}) - 1$:

$$I_X = X^{163} + V^{530855}X + 2.$$

We chose $g = X + V^2$, which we hope is a generator of $\mathbb{F}^*_{3^{12\cdot 163}}$.

To generate an order-r discrete logarithm challenge h, we computed

$$h' = \sum_{i=0}^{162} \left(V^{\lfloor \pi \cdot (3^{12})^{i+1} \rfloor \bmod 3^{12}} \right) X^i$$

[4]More precisely, since C has at most 34 prime factors, each of which is greater than the ten-millionth prime $p = 179424673$, the probability that g is a generator is at least $(1 - \frac{1}{p})^{34} > 0.99999981$.

and then set $h = (h')^{N/r}$. The discrete logarithm $\log_g h \bmod r$ was found to be

$$x = 426395951498279193713291391953449000732592554251132525672039784356054526194343.$$

This can be verified by checking that $h = (g^{N/r})^y$, where $y = x \cdot (N/r)^{-1} \bmod r$.

4.5.2 Experimental results

Our experiments were run on an Intel i7-2600K 3.40 GHz machine (Sandy Bridge), and on an Intel Xeon E5-2650 2.00 GHz machine (Sandy Bridge-EP). The descent strategy was similar to the one used for the $\mathbb{F}_{3^{6\cdot 137}}$ computation.

Relation generation took 0.84 CPU hours (Sandy Bridge, 1 core). The resulting sparse system of linear equations was solved using Magma's multi-threaded parallel version of the Lanczos algorithm; the computation took 852.5 CPU hours (Sandy Bridge, 4 cores).

In the continued-fractions descent stage with $m = 15$, the first degree-81 polynomial yielded 8 irreducible factors of degrees 15, 15, 14, 14, 10, 7, 5, 1, and the second degree-81 polynomial yielded 12 irreducible factors of degrees 12, 10, 9, 9, 9, 8, 6, 6, 6, 4, 1, 1. The computation took 226.7 CPU hours (Sandy Bridge-EP, 16 cores).

Classical descent was used on the 11 polynomials of degree ≥ 8 to obtain polynomials of degree ≤ 7, and then the alternative method in §4.2.5 was used on the 15 polynomials of degree 7 and 30 polynomials of degree 6 to obtain polynomials of degree ≤ 5. These computations took 51.0 CPU hours (Sandy Bridge-EP, 16 cores).

Finally, we used 5-to-3, 4-to-3 and 3-to-2 Gröbner bases descent procedures. The descent was sped up by writing the coefficients of R (see equation (4.17)) in terms of the coefficients of k_1 and k_2; this reduced the number of variables in the resulting bilinear equations from $15m - 3D + 9$ to $9m + 3$. In total, we performed 213 5-to-3 descents, 187 4-to-3 descents, and 11442 3-to-2 descents. These computations took 24.0 CPU hours (Sandy Bridge-EP, 16 cores), 0.8 CPU hours (Sandy Bridge, 4 cores), and 44.8 CPU hours (Sandy Bridge, 4 cores), respectively. The running times of the 2-to-1 descents were incorporated into the running times for the higher-level descents.

4.6 Concluding remarks

We have shown that the finite fields $\mathbb{F}_{3^{6\cdot 1429}}$ offers at most 96 bits of security against the new attacks on the discrete logarithm problem, which is significantly less that the 192 bits of security this field was believed to offer, and solved instances of the discrete logarithm problem in the 1303-bit finite field $\mathbb{F}_{3^{6\cdot 137}}$ and the 1551-bit finite field $\mathbb{F}_{3^{6\cdot 163}}$. These fields are of particular interest in pairing-based cryptography because they contain pairing values for popular bilinear pairings derived from supersingular elliptic curves with embedding degree 6 and are 'general' in that they do not enjoy any Kummer-like properties. Our computations took only 888 CPU hours and 1201 CPU hours, respectively, using modest computer resources despite our implementation being in Magma and far from optimal, unlike the substantial resources (approximately 800,000 CPU hours) that were consumed in [69] for computing a logarithm in the 923-bit field $\mathbb{F}_{3^{6\cdot 97}}$ with the Joux-Lercier algorithm. We emphasize that our upper bound on solving the DLP in $\mathbb{F}_{3^{6\cdot 1429}}$ is quite conservative, and we do not claim that it

is optimal. Nevertheless, the drastic reduction in these security levels and the computational results add further weight to the claim that bilinear pairings derived from the supersingular elliptic curves $E : y^2 = x^3 - x \pm 1$ over $\mathbb{F}_{3^n}$ are unsuitable for pairing-based cryptography.

5 Improved Discrete Logarithms Computations in $\mathbb{F}_{3^{6\cdot 509}}$

5.1 Introduction

In 2014, there were several practical improvements and refinements of algorithms for computing discrete logarithms in finite fields of small characteristic. Most notably, after we had completed the $\mathbb{F}_{3^{6\cdot 137}}$ discrete logarithm computation in February 2014, Granger, Kleinjung and Zumbrägel presented in [60] several practical techniques improving Joux's $L[\frac{1}{4} + o(1)]$ algorithm [77]. These improvements allowed them to compute logarithms in the 4404-bit field $\mathbb{F}_{2^{12\cdot 367}}$ in approximately 52,240 CPU hours, and drastically lowered the estimated time to compute logarithms in the 4892-bit field $\mathbb{F}_{2^{4\cdot 1223}}$ to 2^{59} modular multiplications. One of the techniques described in [60] is the new polynomial representation, originally presented at ECC 2013 by Granger and Zumbrägel [64], and which we subsequently used in [4] and [5] to show its impact on the security of the supersingular elliptic curves $y^2 = x^3 - x \pm 1$ over $\mathbb{F}_{3^{1429}}$ (see §4.3) and to solve instances of the discrete logarithm problem in the 1303-bit field $\mathbb{F}_{3^{6\cdot 137}}$ and the 1551-bit field $\mathbb{F}_{3^{6\cdot 163}}$ (see §4.4 and §4.5).

In April 2014, Granger, Kleinjung and Zumbrägel introduced a new quasi-polynomial time descent method [61] (we call it *powers-of-2* descent), built from the on-the-fly 2-to-1 descent method presented in [57, 58]. This descent method is more practical than the QPA descent of Barbulescu et al. [16], because in the powers-of-2 descent one expresses the logarithm of a polynomial of degree D over $\mathbb{F}_{q^2}$, for instance, in terms of the logarithms of roughly q polynomials of degree at most $D/2$ instead of q^2 in the QPA descent [16]. Kleinjung [90] used this new descent method to compute discrete logarithms in $\mathbb{F}_{2^{1279}}$ in less than four core years.

More recently, Joux and Pierrot [82] presented a more efficient algorithm for computing logarithms of the factor base elements. The new algorithm was used to compute logarithms in the 3796-bit characteristic-three field $\mathbb{F}_{3^{5\cdot 479}}$ in less than 8600 CPU hours. In [82], Joux and Pierrot begin by defining the target field either in Joux's or Granger-Zumbrägel's polynomial representation. Suppose Joux's representation is chosen and, therefore, we want to compute discrete logarithms in $\mathbb{F}_{q^n}$ with $n \leq q + 2$. The field $\mathbb{F}_{q^n}$ is then represented as $\mathbb{F}_q[X]/(I_X)$, where I_X is a degree-n irreducible factor of $h_1(X)X^q - h_0(X)$ with $h_0(X) = \alpha_1 X + \alpha_0$ and $h_1(X) = X^2 + \beta X$ in $\mathbb{F}_q[X]$. Joux and Pierrot [82] exploit the special form of $h_0(X)$ and $h_1(X)$ to accelerate the computation of logarithms of polynomials of degree ≤ 4; the dominant step is the computation of logarithms of degree-3 polynomials, where q linear algebra problems are solved each taking time approximately $q^5/27$ modular additions. The continued-fractions, classical and Gröbner bases descents are all performed over the base field $\mathbb{F}_q$. This is another

idea from [60]. In fact, it is advantageous to first try to descend a target element and all the elements appearing in its descent tree in their respective base fields and not in an extension field, from the outset. This technique was suggested to us by Barbulescu and Gaudry [15] in August 2013, but at that time we did not have a study, as in [60], on the feasibility of the Gröbner bases descent over the base field.

The purpose of this chapter, in joint work with I. Canales, N. Cruz, A. Menezes, T. Oliveira, L. Rivera and F. Rodríguez-Henríquez, is to show that the techniques from [60], [61] and [82] can be used to lower the estimate from Chapter 3 for computing discrete logarithms in the 4841-bit characteristic-three field $\mathbb{F}_{3^{6\cdot 509}}$ from $2^{81.7}M_{q^2}$ to $2^{58.9}M_q$ (where $q = 3^6$ and M_Q denotes the cost of a multiplication in $\mathbb{F}_Q$). As a consequence of this affordable, although challenging, upper bound, we computed discrete logarithms in the order-r subgroup of $\mathbb{F}^*_{3^{6\cdot 509}}$, where $r = (3^{509} - 3^{255} + 1)/7$ is an 804-bit prime, within 220 CPU years. Recall that our main motivation for finding discrete logarithms in $\mathbb{F}_{3^{6\cdot 509}}$ is to attack the elliptic curve discrete logarithm problem in $E(\mathbb{F}_{3^{509}})$, where E is the supersingular elliptic curve $y^2 = x^3 - x + 1$ with $|E(\mathbb{F}_{3^{509}})| = 7r$, which is relevant to the security of Type 1 pairing-based cryptosystems that use bilinear pairings derived from this curve. The elliptic curve discrete logarithm problem in the order-r subgroup of $E(\mathbb{F}_{3^{509}})$ can be efficiently reduced to the discrete logarithm problem in the order-r subgroup of $\mathbb{F}^*_{3^{6\cdot 509}}$.

In addition, we use techniques from [60] to lower the estimate from §4.3 for computing discrete logarithms in the 13590-bit characteristic-three field $\mathbb{F}_{3^{6\cdot 1429}}$ from $2^{95.8}M_{q^2}$ to $2^{78.8}M_{q^2}$ (where $q = 3^6$).

The remainder of the chapter is organized as follows. In §5.2 we review the new techniques from [60], [61] and [82] when incorporated into the DLP algorithm of Joux [77]. Our concrete analysis and experimental results for computing logarithms in $\mathbb{F}_{3^{6\cdot 509}}$ are then presented in §5.3 using the algorithm described in §5.2. In §5.4, we present our new estimates for discrete logarithm computations in $\mathbb{F}_{3^{6\cdot 1429}}$. We make some concluding remarks in §5.5.

5.2 The DLP algorithm of Joux, Granger et al. and Joux-Pierrot

Let $\mathbb{F}_{q^n}$ be a finite field where $n \leq q + 2$. The elements of $\mathbb{F}_{q^n}$ are represented as polynomials of degree at most $n - 1$ over $\mathbb{F}_q$. Let $N = q^n - 1$, and let r be a prime divisor of N. We are interested in the discrete logarithm problem in the order-r subgroup of $\mathbb{F}^*_{q^n}$. More precisely, we are given two elements α, β of order r in $\mathbb{F}^*_{q^n}$ and we wish to find $\log_\alpha \beta$. Let g be an element of order N in $\mathbb{F}^*_{q^n}$. Then $\log_\alpha \beta = (\log_g \beta)/(\log_g \alpha) \bmod r$. Thus, in the remainder of this section we will assume that we need to compute $\log_g h \bmod r$, where h is an element of order r in $\mathbb{F}^*_{q^n}$.

The algorithm we use to solve the DLP in $\mathbb{F}_{q^n}$ is due to Joux [77], with the Joux-Pierrot [82] improvement for computing the logarithms of the factor base elements and methods from Granger et al. [60, 61] for performing the descent steps over the base field $\mathbb{F}_q$.

The algorithm proceeds by precomputing the logarithms of all elements of degree ≤ 3 (§5.2.2–§5.2.3) in $\mathbb{F}_{q^n}$ and some families of degree-4 elements (§5.2.4) [82]. Then, in the descent stage, $\log_g h$ is expressed as a linear combination of logarithms of elements of degree

≤ 3 and elements in the precomputed degree-4 families in $\mathbb{F}_{q^n}$. The descent stage proceeds in several steps, each expressing the logarithm of a degree-D element as a linear combination of the logarithms of elements of degree $\leq m$ for some $m < D$. Five descent methods are used. The continued-fractions, classical and Gröbner bases descents do not differ from how they are described in §3.3.4, §3.6.5, and §3.3.7, respectively, except that here one is directly working over the base field $\mathbb{F}_q$ and not over a proper extension of it. We, therefore, will not recall these descents in this chapter. The degree-4 family-based Groebner bases descent [82] is described in §5.2.4.2 and the powers-of-2 descent [61] in §5.2.5.

5.2.1 Setup

Select elements $\alpha_0, \alpha_1, \beta \in \mathbb{F}_q$ so that the polynomial $h_1(X)X^q - h_0(X)$ has an irreducible factor I_X of degree n in $\mathbb{F}_q[X]$, where $h_0(X) = \alpha_1 X + \alpha_0$ and $h_1(X) = X^2 + \beta X$. Note that

$$X^q \equiv h_0/h_1 \pmod{I_X}.$$

The field $\mathbb{F}_{q^n}$ is represented as $\mathbb{F}_{q^n} = \mathbb{F}_q[X]/(I_X)$ and the elements of $\mathbb{F}_{q^n}$ can be represented as polynomials in $\mathbb{F}_q[X]$ of degree at most $n-1$. Let g be a generator of $\mathbb{F}_{q^n}^*$.

5.2.2 Finding logarithms of quadratic polynomials

Let $\mathcal{B}_2$ be the set of all irreducible polynomials over $\mathbb{F}_q$ of degree ≤ 2 and note that $|\mathcal{B}_2| \approx q^2/2+q$. To compute the logarithms of $\mathcal{B}_2$-elements, we first generate linear relations of these logarithms. Let $a, b \in \mathbb{F}_q$. Substituting

$$Y \mapsto \frac{X^2 + a}{X + b}$$

into the systematic equation

$$Y^q - Y = \prod_{\alpha \in \mathbb{F}_q} (Y - \alpha), \tag{5.1}$$

and then multiplying by $h_1^2(X+b)^{q+1}$ yields

$$\begin{aligned}(h_0^2 + ah_1^2)(X+b) - (X^2+a)(h_0h_1 + bh_1^2) \\ \equiv h_1^2 \cdot (X+b) \cdot \prod_{\alpha \in \mathbb{F}_q} \left(X^2 - \alpha X + a - \alpha b\right) \pmod{I_X}.\end{aligned} \tag{5.2}$$

Note that the left side of (5.2) is a polynomial of degree (at most) 6. If this polynomial is 2-smooth, then taking logarithms of both sides of (5.2) yields a linear relation of the logarithms of $\mathcal{B}_2$-elements.

As first noticed in [60], the left side of (5.2) has a systematic factor $h_1X - h_0$.[1] Then, the cofactor of $h_1X - h_0$ in the left side of (5.2) is of degree 3 and its probability of being

[1] In fact, for any polynomials $k_1, k_2 \in \mathbb{F}_q[X] \backslash \mathbb{F}_q$ of degree at most $D \geq 1$, substituting $Y \mapsto k_1/k_2$ into equation 5.1 and then multiplying by $h_1^D k_2^{q+1}$, gives a left side divisible by $h_1X - h_0$.

2-smooth is $\frac{N_q(1,3)}{q^3} \geq 2/3$. We assume that each pair $(a,b) \in \mathbb{F}_q^2$ generates a distinct relation. Now, since $2/3 \cdot q^2 \gg q^2/2+q$, one expects to obtain sufficiently many linear relations of the logarithms of $\mathcal{B}_2$-elements and, therefore, finds these logarithms by solving a sparse system of linear equations using Wiedemann's algorithm. Thus, after approximately $3/2 \cdot (q^2/2+q)$ trials one expects to obtain $q^2/2+q$ relations. The cost of the relation generation stage is $3/2 \cdot (q^2/2+q) \cdot S_q(1,3)$. The expected cost of the linear algebra is $3/2 \cdot (q^5/4+q^4+q^3) \cdot A_N$, where A_N denotes the cost of an integer addition modulo N.

5.2.3 Finding logarithms of cubic polynomials

Let $\mathcal{B}_3$ be the set of all irreducible polynomials over $\mathbb{F}_q$ of degree 3 and note that $|\mathcal{B}_3| \approx q^3/3$. For $\gamma \in \mathbb{F}_q^*$, we define

$$\mathcal{B}_{3,\gamma} = \{Q = X^3 + aX^2 + bX + \gamma \mid a,b \in \mathbb{F}_q \text{ and } Q \text{ irreducible in } \mathbb{F}_q[X]\}.$$

In this way, $\mathcal{B}_3$ is partitioned into $q-1$ sets each of size roughly $q^2/3$.Thus, to compute the logarithms of the elements in $\mathcal{B}_3$, for every $\gamma \in \mathbb{F}_q^*$ we generate linear relations of logarithms of $\mathcal{B}_{3,\gamma}$-elements and solve the resulting sparse system of linear equations. Let $\gamma \in \mathbb{F}_q^*$, and let $a,b \in \mathbb{F}_q$. Substituting

$$Y \mapsto \frac{X^3 + aX^2 + \gamma}{X^3 + bX + \gamma}$$

into equation (5.1) and then multiplying by $h_1^3(X^3+bX+\gamma)^{q+1}$ yields

$$\begin{aligned}&(h_0^3 + ah_0^2h_1 + \gamma h_1^3)(X^3+bX+\gamma) - (X^3+aX^2+\gamma)(h_0^3 + bh_0h_1^2 + \gamma h_1^3) \qquad (5.3)\\ &\equiv h_1^3 \cdot (X^3+bX+\gamma) \cdot \prod_{\alpha \in \mathbb{F}_q} \big((1-\alpha)X^3 + aX^2 - \alpha bX + (1-\alpha)\gamma\big) \pmod{I_X}.\end{aligned}$$

One can easily see that the left side of (5.3) is a polynomial of degree (at most) 8 divisible by $X(h_0X - h_1)$ and that all the monic irreducible polynomials of degree 3 appearing in the right side are $\mathcal{B}_{3,\gamma}$-elements. If the degree-4 cofactor of $X(h_1X-h_0)$ in the left side of (5.3) is 2-smooth, which happens with probability $\frac{N_q(2,4)}{q^4} \geq 0.41$, then taking logarithms of both sides of (5.3) yields a linear relation of logarithms of degree-1 and degree-2 elements, already known from §5.2.2, and logarithms of $\mathcal{B}_{3,\gamma}$-elements.

Assuming that each pair $(a,b) \in \mathbb{F}_q^2$ generates a distinct relation gives $0.41 \cdot q^2 \gg q^2/3$. This implies that sufficiently many linear relations of the logarithms of $\mathcal{B}_{3,\gamma}$-elements can be obtained and these logarithms are, thereafter, found by solving a sparse system of linear equations using Wiedemann's algorithm. Hence, after approximately $1/0.41 \cdot q^2/3$ trials one expects to obtain $q^2/3$ relations. The cost of the relation generation stage is $1/0.41 \cdot q^2/3 \cdot S_q(2,4)$. The expected cost of the linear algebra is $(q^5/27) \cdot A_N$ since each equation has approximately $q/3$ nonzero terms. The total cost to find logarithms of all $\mathcal{B}_3$-elements is dominated by the $(q-1) \cdot (q^5/27) \cdot A_N$ cost to solve the $q-1$ sparse systems of linear equations.

5.2.4 Finding logarithms of quartic polynomials

The Gröbner bases descent methodology applies for descending degree-4 polynomials. However, similarly to the case of quadratic polynomials in §4.4.2, this is successful for only about 50% of the quartic polynomials over $\mathbb{F}_q$. Using an adapted version of Strategy 4.2, which we call *the Frobenius strategy*, helps to raise this portion to about 58.6%. Unfortunately, Strategy 4.1 does not apply here and so one cannot hope to achieve the higher percentage of §4.4.2. In order to overcome this barrier, Joux and Pierrot [82] extended to quartic elements their idea (§5.2.3) of partitioning the set of polynomials and exploit the special form of h_0 and h_1 to find relations of logarithms of elements in smaller sets and solve the resulting linear algebra problems.

5.2.4.1 Finding logarithms of the factor base quartic polynomials

Let $\mathcal{B}_4$ be the set of all irreducible degree-4 polynomials over $\mathbb{F}_q$ and note that $|\mathcal{B}_4| \approx q^4/4$. For $\gamma \in \mathbb{F}_q^*$, we define

$$\mathcal{B}_{4,\gamma} = \{Q = X^4 + aX^3 + bX^2 + \delta X + \gamma \mid a, b, \delta \in \mathbb{F}_q \text{ and } Q \text{ irreducible in } \mathbb{F}_q[X]\},$$

and then for a fixed $\delta \in \mathbb{F}_q$, we define

$$\mathcal{B}_{4,\gamma,\delta} = \{Q = X^4 + aX^3 + bX^2 + \delta X + \gamma \mid a, b \in \mathbb{F}_q \text{ and } Q \text{ irreducible in } \mathbb{F}_q[X]\}.$$

In this way, $\mathcal{B}_4$ is partitioned into $q-1$ families each of size roughly $q^3/4$ and each family is in turn partitioned into q subfamilies each of size roughly $q^2/4$. Let $\gamma \in \mathbb{F}_q^*$. To compute the logarithms of the elements in $\mathcal{B}_{4,\gamma}$, for every $\delta \in \mathbb{F}_q$ we generate linear relations of logarithms of $\mathcal{B}_{4,\gamma,\delta}$-elements and solve the resulting sparse system of linear equations. Let $\delta \in \mathbb{F}_q$, and let $a, b \in \mathbb{F}_q$. Substituting

$$Y \mapsto \frac{X^4 + aX^2 + \delta X + \gamma}{X^3 + bX^2}$$

into equation (5.1) and then multiplying by $h_1^4(X^3 + bX^2)^{q+1}$ yields

$$\begin{aligned}&(h_0^4 + ah_0^2h_1^2 + \delta h_0h_1^3 + \gamma h_1^4)(X^3 + bX^2) - (X^4 + aX^2 + \delta X + \gamma)(h_0^3h_1 + bh_0^2h_1^2) \qquad (5.4)\\ &\equiv h_1^4 \cdot (X^3 + bX^2) \cdot \prod_{\alpha \in \mathbb{F}_q} \left(X^4 - \alpha X^3 + (a - \alpha b)X^2 + \delta X + \gamma\right) \pmod{I_X}.\end{aligned}$$

As in §5.2.3, one can easily see that the left side of (5.4) is a polynomial of degree (at most) 11 divisible by $X(h_0X - h_1)$ and that all the irreducible polynomials of degree 4 appearing in the right side are $\mathcal{B}_{4,\gamma,\delta}$-elements. If the degree-7 cofactor of $X(h_1X - h_0)$ in the left side of (5.4) is 3-smooth, which happens with probability $\frac{N_q(3,7)}{q^7} \geq 0.24$, then taking logarithms of both sides of (5.4) yields a linear relation of logarithms of elements with degree at most 3, already known from §5.2.2 and §5.2.3, and logarithms of $\mathcal{B}_{4,\gamma,\delta}$-elements.

Recall that the logarithms of 58.6% of the quartic polynomials can be found using the Gröbner bases descent combined with the Frobenius strategy. Assuming that this portion is uniformly distributed over all the quartic polynomials, only $(1 - 0.586) \cdot q^2/4 \leq 0.11 \cdot q^2$

will be involved in the above linear relation generation. We assume furthermore that each pair $(a, b) \in \mathbb{F}_q^2$ generates a distinct relation. Now, since $0.24 \cdot q^2 \gg 0.11 \cdot q^2$, one expects to obtain sufficiently many linear relations of the logarithms of the quartic elements in question and, therefore, finds these logarithms by solving a sparse system of linear equations using Wiedemann's algorithm. Thus, after approximately $0.11/0.24 \cdot q^2$ trials one expects to obtain $0.11 \cdot q^2$ relations. The cost of the relation generation stage is at most $0.11/0.24 \cdot q^2 \cdot S_q(1, 3)$. The expected cost of the linear algebra is at most $0.0013 \cdot q^5 \cdot A_N$.

After the logarithms of all the elements in a first family $\mathcal{B}_{4,\gamma_0}$, $\gamma_0 \in \mathbb{F}_q^*$, are computed, for a second family $\mathcal{B}_{4,\gamma_1}$, $\gamma_1 \in \mathbb{F}_q^*$, one can combine the Gröbner bases descent, the Frobenius strategy and the new descent method described next in §5.2.4.2 to improve the proportion of descending quartics and then reduce the size of the linear equation systems of the $\mathcal{B}_{4,\gamma_1}$-subfamilies. More precisely, since the probability of success of this new descent method is about $1/2$ and when combined with the Frobenius strategy gives a probability of about 0.586, the portion of degree-4 polynomials in $\mathcal{B}_{4,\gamma_1}$ that descend is about $1 - (1 - 0.586)^2 \approx 0.8286$. Therefore, the number of elements involved in the linear equation system of every $\mathcal{B}_{4,\gamma_1}$-subfamily is about $(1 - 0.8286) \cdot q^2/4 \approx 0.043 \cdot q^2$. Applying this procedure recursively for the following families will reduce each time the size of the linear equation system by more than half.

5.2.4.2 Family-based Gröbner bases descent for quartic polynomials

Suppose that some nonzero number (say, s) of families, $\mathcal{B}_{4,\gamma_0}, \ldots, \mathcal{B}_{4,\gamma_{s-1}}$, have the logarithms of all their elements computed and we seek to find the logarithm of an irreducible degree-4 polynomial $Q \in \mathbb{F}_q[X]$ not in those families. Suppose that the Gröbner bases descent and the Frobenius strategy have failed in descending Q. Now, let's consider a first family (say, $\mathcal{B}_{4,\gamma_0}$). Our goal is to find polynomials $k_1 = X^4 + a_2X^2 + a_1X + \gamma_0$ and $k_2 = X^3 + b_2X^2 + b_1X \in \mathbb{F}_q[X]$ such that $Q \mid G$, where

$$G = h_1^4(k_1^q k_2 - k_1 k_2^q) \bmod I_X.$$

In this case, we have

$$G(X) \equiv h_1^4 \cdot (X^3 + b_2X^2 + b_1X) \cdot \prod_{\alpha \in \mathbb{F}_q} \left(X^4 - \alpha X^3 + (a_2 - \alpha b_2)X^2 + (a_1 - \alpha b_1)X + \gamma_0\right) \pmod{I_X} \tag{5.5}$$

as can be seen by making the substitution $Y \mapsto k_1/k_2$ into the systematic equation (5.1) and clearing denominators. It is clear that all the irreducible polynomials of degree 4 appearing in the right side of (5.5) are $\mathcal{B}_{4,\gamma_0}$-elements. Note that

$$\begin{aligned} G = {} & (h_0^4 + a_1h_0^2h_1^2 + a_2h_0h_1^3 + \gamma_0h_1^4)(X^3 + b_2X^2 + b_1X) \\ & - (X^4 + a_2X^2 + a_1X + \gamma_0)(h_0^3h_1 + b_2h_0^2h_1^2 + b_1h_0h_1^3), \end{aligned}$$

which is a degree-11 polynomial divisible by $X(h_1X - h_0)$ and Q. The cofactor of $X \cdot (h_1X - h_0) \cdot Q$ in G is a degree-3 polynomial. Thus, equation (5.5) yields an expression of the logarithm of Q in terms of logarithms of polynomials of degree at most 3 and polynomials in the family $\mathcal{B}_{4,\gamma_0}$. Since these logarithms are all known, we recover the logarithm of Q.

To find polynomials k_1, k_2 such that $Q \mid G$, one proceeds as in the classical Gröbner bases descent, with the same computational cost, where a system of equations is solved using a Gröbner basis finding algorithm.

Similar to what is discussed in §5.2.4.1, this descent method, together with the Frobenius strategy, is successful for only about 58.6% of all irreducible quadratic polynomials not in $\mathcal{B}_{4,\gamma_0}, \ldots, \mathcal{B}_{4,\gamma_{s-1}}$. To ensure descending a quartic polynomial not in $\mathcal{B}_{4,\gamma_0}, \ldots, \mathcal{B}_{4,\gamma_{s-1}}$ with a probability of $1 - 0.414^{s+1}$, one applies a combination of the Frobenius strategy with the Gröbner bases descent and, if this fails, then with the family-based descent, iteratively based on the s precomputed families.

5.2.5 Powers-of-2 descent

Let $Q \in \mathbb{F}_q[X]$ be an irreducible polynomial of degree $2m$, $m > 2$. In [61], one starts by lifting Q to $\mathbb{F}_{q^m}[X]$, where it factors into m irreducible polynomials of degree 2, say, $Q = Q_0 \cdots Q_{m-1}$ with $Q_i \in \mathbb{F}_{q^m}[X]$, $0 \leq i < m$. Note that, in this case, the polynomials Q_i are conjugate in the sense that for every $0 \leq i < m$, there exists $0 \leq j < m$ such that $Q_i = Q_0^{[j]}$ where $Q_0^{[j]}$ denotes the polynomial obtained by raising each coefficient of Q_0 to the power q^j. For simplicity, we shall assume that $Q_i = Q_0^{[i]}$ for all $0 \leq i < m$.

Now, suppose that Q_0, of degree 2, is descended to linear polynomials over $\mathbb{F}_{q^m}$, that is, we have an expression

$$Q_0 \cdot \prod_s F_s = \prod_t G_t,$$

where the F_s and G_t are linear polynomials over $\mathbb{F}_{q^m}$. Then, for every $0 \leq i < m$, we have $Q_i \cdot \prod_s F_s^{[i]} = \prod_t G_t^{[i]}$. This gives

$$Q \cdot \prod_s \left(F_s^{[0]} \cdots F_s^{[m-1]}\right) = \prod_t \left(G_t^{[0]} \cdots G_t^{[m-1]}\right).$$

Since for every pair of indexes (s, t), the products $F_s^{[0]} \cdots F_s^{[m-1]}$ and $G_t^{[0]} \cdots G_t^{[m-1]}$ are nothing more than the respective polynomial norms of the linear polynomials F_s and G_t over $\mathbb{F}_q$ and, therefore, are polynomials in $\mathbb{F}_q[X]$ of degree m, we get an expression of the logarithm of Q in terms of logarithms of polynomials of degree (at most) m.

To descend Q_0 to linear polynomials over $\mathbb{F}_{q^m}$, Granger et al. [61] employ the on-the-fly degree 2-to-1 descent presented in [57, 58]. Let $\mathcal{B}$ be the set of elements $B \in \mathbb{F}_{q^m}$ such that the polynomial $X^{q+1} - BX + B$ factors into linear polynomials over $\mathbb{F}_{q^m}$. Note from [29, 70, 61] that $|\mathcal{B}| \approx q^{m-3}$ and $\mathcal{B}$ can be characterized as the image of $\mathbb{F}_{q^m} \backslash \mathbb{F}_{q^2}$ under the map $u \mapsto (u - u^{q^2})^{q+1}/(u - u^q)^{q^2+1}$. For $B \in \mathcal{B}$ and $a, b, c \in \mathbb{F}_{q^m}$ such that $c \neq ab$, $a^q \neq b$ and $B = \frac{(a^q - b)^{q+1}}{(c-ab)^q}$, one can see by the change of variable $X \to \frac{c-ab}{a^q-b} \cdot X - a$ in $X^{q+1} - BX + B$ that the polynomial $X^{q+1} + aX^q + bX + c$ also factors into linear polynomials over $\mathbb{F}_{q^m}$.

Let $a, b, c \in \mathbb{F}_{q^m}$. We have $h_1 \left(X^{q+1} + aX^q + bX + c\right) \bmod I_X = (X + a)h_0 + (bX + c)h_1$, which is a polynomial of degree at most 3 over $\mathbb{F}_{q^m}$. Thus, if we find $a, b, c \in \mathbb{F}_{q^m}$, with $c \neq ab$, $a^q \neq b$, such that $Q_0 \mid (X + a)h_0 + (bX + c)h_1$ and $\frac{(a^q - b)^{q+1}}{(c-ab)^q} \in \mathcal{B}$, then we succeed in descending Q_0 to linear polynomials over $\mathbb{F}_{q^m}$.

Elements $a, b, c \in \mathbb{F}_{q^m}$ such that $Q_0 \mid (X+a)h_0 + (bX+c)h_1$ can be obtained by finding a basis $\{(u_1, X+u_2), (X+v_1, v_2)\}$ of the lattice

$$L_Q = \{(w_1, w_2) \in \mathbb{F}_{q^m}[X] \times \mathbb{F}_{q^m}[X] \ : \ Q \mid (w_1(X)h_0 - w_2(X)h_1)\}$$

where u_1, u_2, v_1, $v_2 \in \mathbb{F}_{q^m}$, since by choosing $b \in \mathbb{F}_{q^m}$ and letting $a = bu_1 + v_1$ and $c = bu_2 + v_2$, one can easily see that $(X+a, bX+c) \in L_Q$.

To additionally meet the condition $\frac{(a^q-b)^{q+1}}{(c-ab)^q} \in \mathcal{B}$ ($c \neq ab$, $a^q \neq b$), we just need to choose $b \in \mathbb{F}_{q^m}$ such that

$$\mu(b) = \frac{\Big(b^q u_1^q - b + v_1^q\Big)^{q+1}}{\Big(-b^2 u_1 + (u_2 - v_1)b + v_2\Big)^q} \in \mathcal{B},$$

as can be seen by replacing a and c in $\frac{(a^q-b)^{q+1}}{(c-ab)^q}$ by $bu_1 + v_1$ and $bu_2 + v_2$, respectively. Such an element b can be found by iteratively picking an element $B \in \mathcal{B}$ and solving the equation $B = \mu(b)$. One possibility for solving this equation is to write b using m variables over $\mathbb{F}_q$ and obtain a quadratic system of equations that can be solved by finding a Gröbner basis for the ideal it generates. For the asymptotic character of the quasi-polynomial time algorithm presented in [61], finding Gröbner basis, which dominates the running time of this descent method and of exponential complexity in the number of variables, may not be suitable. However, in our practical case of study, this is efficient enough, since we need Gröbner basis for small degrees m only. The probability that an equation $B = \mu(b)$ has solutions is $1/2$. Thus, when $m \geq 4$ ($|\mathcal{B}| \geq q$), the descent of a quadratic polynomial in $\mathbb{F}_{q^m}[X]$ to linear polynomials over $\mathbb{F}_{q^m}$ is always expected to be successful. On the other hand, when $m = 3$, we have $|\mathcal{B}| = 1$ and only 50% of the quadratics over $\mathbb{F}_{q^m}$ are expected to descend.

5.3 Solving the discrete logarithm problem in $\mathbb{F}_{3^{6\cdot509}}$

As in §3.4, we are interested in computing discrete logarithms in the order-r subgroup of $\mathbb{F}^*_{3^{6\cdot509}}$, where $r = (3^{509} - 3^{255} + 1)/7$ is an 804-bit prime. Our approach to computing discrete logarithms in this subgroup is to use the algorithm described in §5.2, whence $q = 3^6$ and $n = 509$.

The DLP instance we solved is described in §5.3.1. The concrete estimates from §5.2 for solving the DLP instance are given in §5.3.2. Our experimental results are presented in §5.3.3. In §5.3.4, we provide some implementation details on the handling of logarithms of degree-4 elements.

5.3.1 Problem instance

Let N denote the order of $\mathbb{F}^*_{3^{6\cdot509}}$. Using the tables from the Cunningham Project [42], we partially factored N as $N = C \cdot p_1^3 \cdot \prod_{i=2}^{21} p_i$, where the p_i are the following primes (and

$r = p_{21}$):

$$p_1 = 2 \quad p_2 = 7 \quad p_3 = 13 \quad p_4 = 1019 \quad p_5 = 7127 \quad p_6 = 21279 \quad p_7 = 54973 \quad p_8 = 97729$$
$$p_9 = 14495303 \quad p_{10} = 39115633 \quad p_{11} = 324927277 \quad p_{12} = 1644550169868135799$$
$$p_{13} = 595618243735721677761652488341 \quad p_{14} = 140832359206526562122960328202050868 7$$
$$p_{15} = 19724128725821325379688781664270351435664812399$$
$$p_{16} = 44582241442151759012783378206529661118466376061093067552696337700081 3$$
$$p_{17} = 74695892089816575593582344546520297133372906987691307120395640648449598307 43$$
$$p_{18} = 31630163990546614532161673567136353963341968419052700279077711962913108713 46$$
$$2585889709129767869836324 9$$
$$p_{19} = 23214049468525744074633683839531277461402596152346780076104082330578039306 50$$
$$5889278179026503395282728989110957696917932365850935308873749615837273$$
$$p_{20} = 44816150161927977920647360929674418409626760138935782894188283534375872109 56$$
$$6942820514222439432850646355157098126051586360410836283973747433357183937067$$
$$4423738520 3$$
$$p_{21} = 10223994620258685240980988741809302145715061249525570661473300332752627908 15$$
$$6368783078274830574618706026498586928352444181958959275099808618631525078106$$
$$7131293823177124077445718802216415539934838376431091001197641295264650596195$$
$$201747790167311,$$

and

$$C = (T^2 + T + 1)/13, \text{ with } T = 3^{509},$$

is a 1610-bit composite number.

We verified that $\gcd(C, N/C) = 1$ and that C is not divisible by any of the first 10^7 primes. Consequently, as mentioned in §4.5.1, if an element g is selected uniformly at random from $\mathbb{F}^*_{3^{6\cdot 509}}$, and g satisfies $g^{N/p_i} \neq 1$ for $1 \leq i \leq 21$, then g is a generator with very high probability.

The field $\mathbb{F}_{3^6}$ is represented as $\mathbb{F}_3[u]/(u^6 + 2u^4 + u^2 + 2u + 2)$ and u is a generator of $\mathbb{F}_{3^6}$. The field $\mathbb{F}_{3^{6\cdot 509}}$ is represented as $\mathbb{F}_{3^6}[X]/(I_X)$, where I_X is the degree-509 irreducible factor of $h_1(X)X^q - h_0(X)$ with $h_0(X) = u^{316}X + u^{135}$ and $h_1(X) = X^2 + u^{424}X$.

We chose the generator $g = X + u^2$ of $\mathbb{F}^*_{3^{6\cdot 509}}$. To generate an order-$r$ discrete logarithm challenge h, we computed

$$h' = \sum_{i=0}^{508} \left(V^{\lfloor \pi \cdot (3^6)^{i+1} \rfloor \bmod 3^6} \right) X^i$$

and then set $h = (h')^{N/r}$. The discrete logarithm was found to be

$$x = 14918739986031826636021663369329699337745628189156992132531381787777037864304930 6$$
$$6480809525652983536765790074515932016074643909955366015387428992219846518967940 08$$
$$8550218976628414869296477962794457122205313359696590735777748798909231235385145 6.$$

This can be verified by checking that $h = (g^{N/r})^y$, where $y = x \cdot (N/r)^{-1} \bmod r$.

5.3.2 Estimates

The dominant step in the computation of the logarithms of elements in $\mathcal{B}_2$, $\mathcal{B}_3$ and $\mathcal{B}_{4,\gamma}$ for all $\gamma \in S$, where S is some subset of $\mathbb{F}_q^*$, is the solving of the $(q-1)$ linear algebra problems arising from the computation of logarithms of $\mathcal{B}_3$-elements, each with cost approximately $q^5/27\,A_r$. The continued-fractions, classical, Gröbner bases and powers-of-2 descents are used to express the discrete logarithm of the challenge element h in terms of logarithms of elements in the factor base $\mathcal{B}_2$, $\mathcal{B}_3$, $\mathcal{B}_{4,\gamma}$, $\gamma \in S$.

The new cost estimates are presented in Table 5.1. We used the estimates for smoothness testing from [59], and the 'bottom-top' approach from [60] for estimating the cost of Gröbner bases and powers-of-2 descents from degree 15 to degree 4. We assume that 2^{27} multiplications in $\mathbb{F}_{3^6}$ can be performed in 1 second; we achieved this performance using a look-up table approach. The timings for Gröbner bases descent and $\mathbb{F}_{3^6}$ multiplications were obtained on an Intel i7-3930K 3.2 GHz CPU core. In a non-optimized C implementation, we have observed an A_r cost of 43 clock cycles, where lazy reduction is used to amortize the cost of a modular reduction among many integer additions. This yields the cost ratio $A_r/M_q \approx 2$.

The main effect of the new improvements is the removal of the QPA descent stage from the estimates in §3.4. The overall running time is $2^{58.9}M_q$, a significant improvement over the $2^{81.7}M_{q^2}$ estimate from §3.4. In particular, assuming the availability of processors that can perform 2^{27} $\mathbb{F}_{3^6}$-multiplications per second, the estimated running time is approximately 127 CPU years — this is a feasible computation that we performed and the results are presented in §5.3.3 (and §5.3.4).

Finding logarithms of polynomials of degree ≤ 4		
Linear algebra	$2^{52.3}A_r$	$2^{53.3}M_q$
Descent		
Continued-fractions (254 to 40)	$2^{56.9}M_q$	$2^{56.9}M_q$
Classical (40 to 21)	$12.7 \times 2^{54.2}M_q$	$2^{57.9}M_q$
Classical (21 to 15)	$159 \times 2^{49.4}M_q$	$2^{56.7}M_q$
Gröbner bases, powers-of-2 (15 to 4)	1924×8249 seconds	$2^{50.9}M_q$

Table 5.1: Estimated costs of the main steps for computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 509}}$ ($q = 3^6$). A_r and M_q denote the costs of an addition modulo the 804-bit prime $r = (3^{509}-3^{255}+1)/7$ and a multiplication in $\mathbb{F}_{3^6}$. We use the cost ratio $A_r/M_q = 2$, and also assume that 2^{27} multiplications in $\mathbb{F}_{3^6}$ can be performed in 1 second.

5.3.3 Experimental results

The computation described in this section was done using clusters from Cinvestav's ABACUS supercomputer [1] (5103 CPU cores used), Cinvestav's Computer Science Department (290 CPU cores used), and University of Waterloo's Faculty of Mathematics (120 CPU cores used). Table 5.2 shows the CPU time consumed at each stage of the computation.

Computation stage	CPU time (years)	CPU frequency (GHz)
Finding logarithms of quadratic polynomials		
Relation generation	0.01	3.20
Linear algebra	0.50	2.40
Finding logarithms of cubic polynomials		
Relation generation	0.15	3.20
Linear algebra	43.88	2.60
Finding logarithms of quartic polynomials		
Relation generation	4.07	2.60
Linear algebra	96.02	2.60
Descent		
Continued-fractions (254 to 40)	51.71	2.87
Classical (40 to 21)	9.99	2.66
Classical (21 to 15)	10.24	2.66
Gröbner bases, powers-of-2 (15 to 4)	6.27	3.00
Total CPU time (years)	222.81	

Table 5.2: CPU times of each stage of the computation of discrete logarithms in $\mathbb{F}_{3^{6\cdot 509}}$.

5.3.3.1 Computing of the logarithms of elements in $\mathcal{B}_2$, $\mathcal{B}_3$ and $\mathcal{B}_{4,u^i}$, $i \in [0, 28]$

Finding logarithms of quadratic polynomials. The factor base $\mathcal{B}_2$ has size $266,086 \approx 2^{18}$. Relation generation took 2.4 CPU hours using Magma on an Intel i7-3930K 3.20GHz CPU core. The resulting sparse system of linear equations was solved using our C implementation of Wiedemann's algorithm; the computation took $4,320$ CPU hours on an Intel Xeon E5-2658 v2 2.40 GHz CPU core.

Finding logarithms of cubic polynomials. For every $\gamma \in \mathbb{F}_q^*$, $\mathcal{B}_{3,\gamma}$ has size exactly

$$\frac{I_q(3)}{q-1} = \frac{q^2+q}{3} = 177,390 \approx 2^{17.44}.$$

The total relation generation running time is $1,232$ CPU hours using Magma on an Intel i7-3930K 3.20GHz CPU core. The resulting 728 sparse systems of linear equations were solved using our C implementation of Wiedemann's algorithm in $379,142$ CPU hours on an Intel Xeon E5-2697 v3 2.60 GHz CPU core.

Finding logarithms of quartic polynomials in $\mathcal{B}_{4,u^i}, i \in [0, 28]$. In this phase, we could apply the trick described in §5.2.4.1, namely to iteratively keep reducing in each new quartic family the size of linear algebra systems by a factor less than $1/2$. However, because of our limited memory resources and the delay this could cause when generating the linear systems in contrast to the ability of solving a large number of equal size linear algebra problems

in a shorter real time, we preferred to not take advantage of this trick. Therefore, for each $i \in [0, 28]$ and $\delta \in \mathbb{F}_q$, the subfamily $\mathcal{B}_{4,u^i,\delta}$ has size roughly $(1-0.586)\cdot q^2/4 \approx 55,050 \approx 2^{15.75}$. The total relation generation running time was 35,118 CPU hours using Magma on an Intel Xeon E5-2650 v2 2.60GHz CPU core. The resulting $29 \times 729 = 21,141$ sparse systems of linear equations were solved using our C implementation of Wiedemann's algorithm in 829,573 CPU hours on an Intel Xeon E5-2697 v3 2.60 GHz CPU core.

5.3.3.2 Continued-fractions and classical descents

In the continued-fractions descent stage, the two degree-254 polynomials yielded 22 irreducible factors with 2 of degree 40, 1 of degree 39, 1 of degree 38, 1 of degree 37, and 7 of degree in the interval [22, 35]. The computation took 446,768 CPU hours on CPU cores with average frequency 2.87 GHz (270 machines of different frequencies were used in this stage).

In the first classical descent phase, 255 polynomials of degree at most 21 are obtained from the 12 polynomials of degree ≥ 22. These computations took 86,323 CPU hours on CPU cores with average frequency 2.66 GHz (390 machines of different frequencies were used in the two phases of the classical descent).

The second classical descent phase was used on the 84 polynomials of degree ≥ 16 coming from the first phase, to obtain polynomials of degree ≤ 15. These computations took 88,452 CPU hours on CPU cores with average frequency 2.66 GHz.

The number of obtained polynomials of each degree in the interval [5, 15] from the continued-fractions and classical descents is shown in Table 5.3

Complete details on these continued-fractions and classical descent stages can be found in Canales's Master's thesis [33].

Degree	5	6	7	8	9	10	11	12	13	14	15
Number of polynomials	101	107	94	98	92	116	137	123	155	173	213

Table 5.3: Number of polynomials of each degree in the interval [5, 15] obtained after the continued-fractions and classical descents (with total number 1409).

5.3.3.3 Gröbner bases-like and powers-of-2-like descents

After this stage, the 1409 polynomials of degree in the interval [5, 15] resulting from the continued-fractions and classical descents should have their logarithm expressed in terms of logarithms of elements in the factor base, namely, in $\mathcal{B}_2$, $\mathcal{B}_3$ and $\mathcal{B}_{4,u^i}$, $i \in [0, 28]$.

For a degree-D element, $D \in [5, 15]$, the Gröbner bases descent writes the logarithm of this element as a linear combination of logarithms of elements of degree at most d where $d = \lfloor D/2 \rfloor + 2$. This is the best that can be done because of the condition [60, Condition (10)]

$$q^{2d+1-D} \gg q^{3d-D}/N_q(d, 3d-D).$$

The Gröbner bases descent is used on polynomials of odd degree; for those of even degree the powers-of-2 descent is employed (except for degree-14 polynomials, see below) because of its more aggressive descent character. Indeed, a polynomial of degree $2d$, $d > 2$, is related

with polynomials of degree d using the powers-of-2 descent instead of polynomials of degree $d+2$ when using the Gröbner bases descent. The degree-4 polynomials which are not in the factor base are descended using the classical or family-based Gröbner bases descent combined with the Frobenius strategy.

As mentioned in §5.2.5, the powers-of-2 descent is successful for only 50% of the degree-6 polynomials. For the remainder, we used a hybrid Gröbner bases-powers-of-2 descent. In this hybrid descent, a degree-6 polynomial is lifted to the quadratic extension of $\mathbb{F}_{3^6}$, where it splits into two polynomials of degree 3. Over the field $\mathbb{F}_{3^{2\cdot 6}}$, we adapted the Gröbner bases descent in §3.3.7 and used it to perform a 3-to-2 descent on one of the two degree-3 polynomials over $\mathbb{F}_{3^{2\cdot 6}}$. Then, using the polynomial norm as in §5.2.5, we got the logarithm of the degree-6 polynomial expressed in term of logarithms of polynomials of degree (at most) 4. This strategy allowed us to avoid the more costly 6-to-5 and then 5-to-4 Gröbner bases descent steps.

We also employed the hybrid descent on the degree-14 polynomials to have 14-to-8 descents on these polynomials instead of powers-of-2 14-to-7 descents. In fact, a complete descent is more costly on a degree-7 polynomial than on a degree-8 polynomial since in the former two Gröbner bases descent stages, 7-to-5 then 5-to-4, are needed while only one powers-of-2 8-to-4 descent stage is needed in the latter.

Table 5.4 shows the average times for computing the discrete logarithms of all polynomials of degrees in the interval $[5, 15]$ that arise from the continued-fractions and classical descents stages using the fastest method, as described in the present section. These times come from the analysis and experiments outlined in §5.3.4.

Degree	5	6	7	8	9	10	11	12	13	14	15
Average time (s)	$2^{10.21}$	$2^{10.29}$	$2^{17.30}$	$2^{12.16}$	$2^{18.14}$	$2^{19.92}$	$2^{24.57}$	$2^{20.00}$	$2^{24.78}$	$2^{20.91}$	$2^{26.48}$

Table 5.4: Average CPU times in seconds to obtain the complete descent and the discrete logarithms of all the polynomials resulting from the continued-fractions and classical descents, of degrees in the interval $[5, 15]$, assuming that an Intel Xeon E5-2658 v2 2.40 GHz machine with 256 gigabytes of RAM is used.

5.3.4 Some implementation details on the computation of logarithms of degree-4 elements

Theorem 4 of [91] shows that the expected number of degree-d irreducible factors of a randomly selected degree-n polynomial over $\mathbb{F}_q$ is approximately $1/d$. Using this result, we computed the average number of degree-4 elements that are obtained after a descent of a polynomial of degree in the interval $[5, 15]$. We then used Table 5.3 to estimate the expected number of degree-4 irreducible polynomials that result from the total descent steps; these estimates are shown in Table 5.5.

To compute the discrete logarithm of a polynomial in $\mathcal{B}_{4,u^i}$, $i > 28$, we need to express it, using the methods discussed in §5.2.4.2, in terms of logarithms of polynomials of degree

Degree	5	6	7	8	9	10	11	12	13	14	15
Number of degree-4 polynomials	$2^{14.18}$	$2^{14.26}$	$2^{21.27}$	$2^{16.13}$	$2^{22.11}$	$2^{23.88}$	$2^{28.54}$	$2^{23.97}$	$2^{28.74}$	$2^{24.88}$	$2^{30.45}$

Table 5.5: Expected number of degree-4 polynomials resulting from all the Gröbner bases and powers-of-2 descent steps for each degree in the interval $[5, 15]$.

at most 3, and possibly of polynomials in $\mathcal{B}_{4,u^i}$ for some $i \in [0, 28]$. Thus, we first try to have a descent to polynomials of degree at most 3. If this fails, we try to have a descent based on the family $\mathcal{B}_{4,1}$. If this fails too, we try to have a descent based on the family $\mathcal{B}_{4,u}$. We keep trying until we reach the family $\mathcal{B}_{4,u^{28}}$. Table 5.6 shows for every family from $\mathcal{B}_3$, $\mathcal{B}_{4,u^i}$, $i \in [0, 28]$, the inverse of the probability for a random irreducible degree-4 polynomial to descend based on that family.

$\mathcal{B}_3$	$2^{00.8297}$	$\mathcal{B}_{4,u^0}$	$2^{02.1020}$	$\mathcal{B}_{4,u^1}$	$2^{03.3742}$	$\mathcal{B}_{4,u^2}$	$2^{04.6466}$	$\mathcal{B}_{4,u^3}$	$2^{05.9189}$
$\mathcal{B}_{4,u^4}$	$2^{07.1912}$	$\mathcal{B}_{4,u^5}$	$2^{08.4635}$	$\mathcal{B}_{4,u^6}$	$2^{09.7358}$	$\mathcal{B}_{4,u^7}$	$2^{11.0081}$	$\mathcal{B}_{4,u^8}$	$2^{12.2803}$
$\mathcal{B}_{4,u^9}$	$2^{13.5526}$	$\mathcal{B}_{4,u^{10}}$	$2^{14.8250}$	$\mathcal{B}_{4,u^{11}}$	$2^{16.0972}$	$\mathcal{B}_{4,u^{12}}$	$2^{17.3695}$	$\mathcal{B}_{4,u^{13}}$	$2^{18.6418}$
$\mathcal{B}_{4,u^{14}}$	$2^{19.9141}$	$\mathcal{B}_{4,u^{15}}$	$2^{21.1864}$	$\mathcal{B}_{4,u^{16}}$	$2^{22.4587}$	$\mathcal{B}_{4,u^{17}}$	$2^{23.7310}$	$\mathcal{B}_{4,u^{18}}$	$2^{25.0033}$
$\mathcal{B}_{4,u^{19}}$	$2^{26.2756}$	$\mathcal{B}_{4,u^{20}}$	$2^{27.5479}$	$\mathcal{B}_{4,u^{21}}$	$2^{28.8202}$	$\mathcal{B}_{4,u^{22}}$	$2^{30.0925}$	$\mathcal{B}_{4,u^{23}}$	$2^{31.3648}$
$\mathcal{B}_{4,u^{24}}$	$2^{32.6371}$	$\mathcal{B}_{4,u^{25}}$	$2^{33.9094}$	$\mathcal{B}_{4,u^{26}}$	$2^{35.1817}$	$\mathcal{B}_{4,u^{27}}$	$2^{36.4540}$	$\mathcal{B}_{4,u^{28}}$	$2^{37.7263}$

Table 5.6: For every family from $\mathcal{B}_3$, $\mathcal{B}_{4,u^i}$, $i \in [0, 28]$, the inverse of the probability for a random irreducible degree-4 polynomial to descend based on that family.

Since we do not have a native implementation of Gröbner bases finding algorithms, we used Magma's implementation of Faugère's F4 algorithm [47]. Our Magma script for the descent phase frequently needs to read the logarithm of a polynomial in $\mathcal{B}_{4,u^i}$, $i \in [0, 28]$ with frequencies shown in Table 5.6.

The files of logarithms for each of the 29 quartic families has size 20.4 gigabytes, and the files of logarithms for the cubic polynomials has size 26.4 gigabytes; the total size of the logarithm files is 618 gigabytes. On one hand, our machines have only 256 gigabytes of RAM. Thus, some families must be stored in the hard disk (HD), which is much slower to access than the virtual memory (VM). Since many copies of the Magma code will be executed in parallel, each of which will be accessing the same logarithm files, the memory accesses have to be carefully scheduled to avoid traffic congestion. In addition, we had to deal with some restrictions on Magma's file reading capabilities (for example, whether the files are stored in hexadecimal encoding or binary encoding) and with limits on the total number of open files permitted on Linux.

Parameterizing the degree-4 polynomials. For practical purpose in the logarithm reading process, any degree-4 polynomial (irreducible or not) will be characterized by three parameters in order to get a natural partitioning.

For $a \in [0, 728]$ and $f \in \mathbb{F}_{3^6}$, we denote by f^a the natural field exponentiation if $a \in [0, 727]$, and 0 if $a = 728$. Let $a, b, j \in [0, 728]$ and $i \in [0, 727]$. Let $Q = X^4 + u^a X^3 + u^b X^2 + u^j X + u^i$ be a polynomial over $\mathbb{F}_{3^6}$. We define the parameters of Q as:

- i, its family parameter.
- j, its subfamily parameter.
- $pos = (a + 1 \mod 3^6) \cdot 3^6 + (b + 2 \mod 3^6)$, its position in the ordered (with respect to pos and starting from 1) list of all polynomials (irreducibles and reducibles) having i as family parameter and j as subfamily parameter.

We denote the parametrized polynomial Q by $Q_{i,j,pos}$.

The Magma-only logarithm reading process. The logarithms of all the elements in the families $\mathcal{B}_{4,u^i}$, $i \in [0, 28]$ are stored in several different files and the repartition is done in such a way that all and only the logarithms of the polynomials with same family parameter i and same subfamily j parameter are in one file, with path "4_i/log/j.log". All the logarithms occupy the same number of bytes, denoted M, which is the length of the modulus r (202 bytes in text hexadecimal encoding or 101 in binary encoding). [2] Thus, to retrieve the logarithm of an irreducible polynomial, say, $Q_{i,j,pos}$, we only need to go to the file "4_i/log/j.log" at the position of the first byte of the logarithm and read M bytes. But how to get the position of the first byte?

For that, we set index files as "4_i/idx/j.idx", for storing all and only the indexes of the degree-4 polynomials with same family parameter i and same subfamily parameter j, where the index of a reducible polynomial is always 0. For the irreducible polynomials, the one with least third parameter has index 1, the second irreducible in the list has index 2, and so forth. Here also, all the indices occupy the same number of bytes, denoted I, which is the length of $729^2/4 + 729$ (5 bytes in hexadecimal encoding or 3 in binary encoding). Therefore, for an irreducible degree-4 polynomial $Q_{i,j,pos}$ the position of the first byte of its logarithm is obtained by going to the file "4_i/idx/j.idx" at the byte $(pos - 1)I$, reading I bytes, then getting the index e, and finally computing $(e - 1)M$ to retrieve the logarithm first byte position. This being so, it is now easy to describe how Magma proceeds to read logarithms.

At start up, Magma initializes two two-dimensional arrays, Idx4[29][729] and Log4[29][729], of dimension 29x729 each, where the cell Log4[i+1][j+1], for instance, stores a file structure pointing to the file "4_i/log/j.log". The array Idx4 stores file structures for the index files in the same way. At logarithm request, with parameters i, j, pos, Magma asks for the offset of the file position indicator of Idx4[i+1][j+1] and moves it to the offset $(pos - 1)I$, reads I

[2] In the text hexadecimal encoding, 2 bytes are used to encode integers from 0 to $2^8 - 1$, while this is only one byte in the binary encoding.

bytes and convert them to an integer to get an index e. Then it asks for the offset of the file position indicator of Log4[i+1][j+1] and moves it to the offset $(e-1)M$, reads M bytes and convert them to an integer to get the sought logarithm.

Regarding the storage of the indexes and logarithms, we have different options that are functions of how are they stored, in hexadecimal or binary encoding, and where, in the HD or in the VM. The advantage of a binary encoding is that the sizes of the files are half the sizes in the hexadecimal encoding, then allowing to place more families $\mathcal{B}_{4,u^i}$, $i \in [0, 28]$, into the VM (fast memory but with limited size). Unfortunately, the unique operation that Magma can perform with a binary file is the reading of the entirety of its contents, in one call. This means that the strategy we described above will be possible only with text files (in hexadecimal encoding in our case). This is the reason why we need a hybrid Magma-C implementation to handle binary files with more flexibility and freedom.

The Magma-C logarithm reading process. Suppose we are running our Magma script and at one point the logarithm of a polynomial $Q_{i,j,pos}$ in $\mathcal{B}_{4,u^i}$, $i \in [0, 28]$, is needed. Then Magma, by its Pipe function, sends a shell command that executes the C program "bin_rd_509_magma.c" with arguments the parameters of Q, and waits to receive the requested logarithm in the same Pipe call. See the following Magma line code.

```
>log4 := Pipe("./bin_rd_509_magma.out", "i j pos");
```

However, calling the program "bin_rd_509_magma.c" assumes that another program is already running. This is the program "bin_rd_509.c", which is simply the one that retrieves the sought logarithm. We can see here that the program "bin_rd_509_magma.c" in fact acts just as an interface between the Magma process and the program "bin_rd_509.c". When "bin_rd_509_magma.c" is invoked with arguments "*i j pos*", it sends these to the program "bin_rd_509.c" and waits for the logarithm. After receiving the logarithm, it outputs it to Magma by a printing to the standard output, in hexadecimal representation.

This process requires an efficient tool for the C programs to communicate. We used the System V shared memory, which lets multiple unrelated processes attach a segment of physical memory to their virtual address spaces. In our case, "bin_rd_509.c" creates a shared memory segment with a specific key and attaches it to its virtual address space. Then, every time "bin_rd_509_magma.c" is called by Magma, it attaches the created segment to its virtual address space using the same key. Actually, "bin_rd_509.c" creates four shared memory segments, two for receiving the parameters from "bin_rd_509_magma.c" and two others for sending the logarithm to "bin_rd_509_magma.c". We need two segments for each action because one of them is employed as a semaphore to prevent inconsistencies and collisions. To read logarithms, "bin_rd_509.c" proceeds exactly in the same way as in the Magma-only logarithm reading process.

Note that we can consider the degree-3 polynomials as an extra family in which a polynomial $Q = X^3 + u^a X^2 + u^b X + u^j$, where $a, b \in [0, 728]$ and $j \in [0, 727]$, has parameters (-1, j, pos).

- j, the subfamily parameter.
- $pos = (a + 1 \mod 3^6) \cdot 3^6 + (b + 2 \mod 3^6)$, the position in the ordered (with respect

to *pos* and starting from 1) list of all degree-3 polynomials (irreducibles and reducibles) having j as subfamily parameter.

Therefore, reading the logarithm of a degree-3 irreducible polynomial is done exactly as for the logarithm of any element in a given family $\mathcal{B}_{4,u^i}$, $i \in [0, 28]$.

The hybrid Magma-C implementation works without any problem when the binary files are located in the HD or in the VM. In contrast, the Magma-only implementation works correctly only when the files are in the VM (otherwise the data transfer traffic easily gets congested).

Experimental running times. The following experiments were run on a 20-core Intel Xeon E5-2658 v2 2.40 GHz machine with 256 gigabytes of RAM.

- Size of a quartic family in hexadecimal: 20.4 gigabytes.
- Size of the cubic family in hexadecimal: 26.4 gigabytes.

We have three cases, each with its overall running times for reading one logarithm.

- Logarithms and indexes are in hexadecimal in the VM (Magma-only):
  Cpu time: 0.000018 seconds / Real time: 0.0000185 seconds
- Logarithms and indexes are in binary in the VM (Magma-C):
  Cpu time: 0.00186 seconds / Real time: 0.0019 seconds
- Logarithms and indexes are in binary in the HD (Magma-C):
  Cpu time: 0.004 seconds / Real time: 0.024 seconds

These times show, and that is what is experimentally observed, that the best option is to put as many families in hexadecimal encoding as possible in the VM. In the 20-core Intel Xeon E5-2658 v2 2.40 GHz machine with 256 gigabytes of RAM, this is done for the family of irreducible degree-3 polynomials and the families $\mathcal{B}_{4,1}$, $\mathcal{B}_{4,u}$, ..., $\mathcal{B}_{4,u^9}$, and the rest left in the HD in binary. The times for a complete descent of a random irreducible degree-4 polynomial are:

- Cpu time: 0.0614s / Real time: 0.0640s

5.4 Computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 1429}}$

As in §4.3, we are interested in computing discrete logarithms in the order-r subgroup of $\mathbb{F}^*_{3^{6\cdot 1429}}$, where $r = (3^{1429} - 3^{715} + 1)/7622150170693$ is a 2223-bit prime. To accomplish this, we embed $\mathbb{F}_{3^{6\cdot 1429}}$ in its quadratic extension $\mathbb{F}_{3^{12\cdot 1429}}$. Let $q = 3^6$ and $k = 2$. The field $\mathbb{F}_{3^{12\cdot 1429}}$ is represented as $\mathbb{F}_{q^2}[X]/(I_X)$, where I_X is a monic degree-1429 irreducible factor of $h_1(X^q) \cdot X - h_0(X^q)$ with $h_0, h_1 \in \mathbb{F}_{q^2}[X]$ and $\max(\deg h_0, \deg h_1) = 2$.

The techniques from [60] employed to improve the estimates of [4] are the following:

1. Since logarithms are actually sought in the field $\mathbb{F}_{3^{6\cdot 1429}}$, the continued fractions and classical descent stages are performed over $\mathbb{F}_q$ (and not $\mathbb{F}_{q^2}$).

2. In the final classical descent stage to degree 11, one permits irreducible factors over $\mathbb{F}_q$ of even degree up to 22; any factors of degree $2t \geq 12$ that are obtained can be written as a product of two degree-t irreducible polynomials over $\mathbb{F}_{q^2}$.

3. The number of irreducible factors of an m-smooth degree-t polynomial is estimated as t/m.

4. The smoothness testing estimates from Appendix B of [59] were used (see §2.5).

The remaining steps of the algorithm, namely finding logarithms of linear polynomial, finding logarithms of irreducible quadratic polynomials, QPA descent, and Gröbner bases descent, are as described in §4.3.

The new cost estimates are presented in Table 5.7. The main effect of the techniques from [60] is the removal of one QPA descent stage. The overall running time is $2^{78.8}M_{q^2}$, a significant improvement over the $2^{95.8}M_{q^2}$ estimate from §4.3.

Finding logarithms of linear polynomials		
Relation generation	$2^{28.6}M_{q^2}$	$2^{28.6}M_{q^2}$
Linear algebra	$2^{47.5}A_r$	$2^{49.5}M_{q^2}$
Finding logarithms of irreducible quadratic polynomials		
Relation generation	$3^{12} \times 2^{37.6}M_{q^2}$	$2^{56.6}M_{q^2}$
Linear algebra	$3^{12} \times 2^{47.5}A_r$	$2^{68.5}M_{q^2}$
Descent		
Continued-fractions (714 to 88)	$2^{77.6}M_q$	$2^{77.6}M_q$
Classical (88 to 29)	$16.2 \times 2^{73.5}M_q$	$2^{77.5}M_q$
Classical (29 to 11)	$267.3 \times 2^{70.8}M_q$	$2^{78.9}M_q$
QPA (11 to 7)	$2^{13.9} \times (2^{44.4}M_{q^2} + 2^{47.5}A_r)$	$2^{63.4}M_{q^2}$
Gröbner bases (7 to 4)	$2^{35.2} \times$ (76.9 seconds)	$2^{67.5}M_{q^2}$
Gröbner bases (4 to 3)	$2^{44.7} \times$ (0.03135 seconds)	$2^{65.7}M_{q^2}$
Gröbner bases (3 to 2)	$2^{54.2} \times$ (0.002532 seconds)	$2^{71.6}M_{q^2}$

Table 5.7: Estimated costs of the main steps for computing discrete logarithms in $\mathbb{F}_{3^{12\cdot1429}}$ ($q = 3^6$). A_r, M_q, and M_{q^2} denote the costs of an addition modulo the 2223-bit prime r, a multiplication in $\mathbb{F}_{3^6}$, and a multiplication in $\mathbb{F}_{3^{12}}$. We use the cost ratio $A_r/M_{q^2} = 4$, and also assume that 2^{26} (resp. 2^{27}) multiplications in $\mathbb{F}_{3^{12}}$ (resp. $\mathbb{F}_{3^6}$) can be performed in 1 second (cf. §5.3.2).

5.5 Concluding remarks

We used the techniques introduced by Granger et al. [60, 61] and Joux and Pierrot [82] to improve our previous estimates for computing discrete logarithms in the finite fields $\mathbb{F}_{3^{6\cdot509}}$ and $\mathbb{F}_{3^{6\cdot1429}}$. With the new estimates, we solved a DLP instance in the order-r subgroup of $\mathbb{F}^*_{3^{6\cdot509}}$, with $r = (3^{509} - 3^{255} + 1)/7$ an 804-bit prime. The computations took about 220 CPU years.

An important open question is whether the different improvements on the new attacks can be taken further so that they can actually be utilized to effectively solve instances of the discrete logarithm problem in the order-r subgroup of $\mathbb{F}^*_{3^{6\cdot 1429}}$, where $r = (3^{1429} - 3^{715} + 1)/7622150170693$ is a 2223-bit prime. For this to happen, any attempt at removing the remaining QPA descent stage may lead to a fruitful outcome.

6 Another Work: Square Roots in Even-Degree Extensions of Finite Fields

6.1 Introduction

Computing square roots in finite fields is a classical number theory problem that has been addressed by mathematicians over centuries. These days, it is especially relevant for elliptic curve cryptography, where hashing an arbitrary message to a random point in a given elliptic curve [31], decompression of an elliptic curve point [103, 72, 20] and counting points on elliptic curves [112, 12], are some of its most important applications. These cryptographic operations, performed in a large number of pairing-based protocols defined over popular choices of pairing-friendly elliptic curves such as the Barreto-Naehrig (BN) [20, 56], the Kachisa-Schaefer-Scott and the Barreto-Lynn-Scott [19] elliptic curves, often require computing square roots in either quadratic or cubic extensions of finite fields.

Let $q = p^m$, with p an odd prime and m a nonzero positive integer. The task of computing a square root of an arbitrary element $a \in \mathbb{F}_q$ consists in finding a second element $b \in \mathbb{F}_q$, if it exists, such that $b^2 = a$. According to Euler's criterion, also known as the quadratic residuosity test, a square root of an element $a \in \mathbb{F}_q^*$ exists in $\mathbb{F}_q$ if and only if $a^{\frac{q-1}{2}} = 1$. We denote by $\chi_q(a)$ the value of $a^{\frac{q-1}{2}}$. If $\chi_q(a) = 1$, we say that the element a is a *quadratic residue* (QR) in $\mathbb{F}_q$. It is known that in $\mathbb{F}_q^*$ there exist exactly $(q-1)/2$ quadratic residues.

Two non-deterministic classical algorithms for computing square roots in finite fields are the Tonelli-Shanks [120] and the Cipolla-Lehmer [38] algorithms. [1] However, finding square roots can be achieved more easily using specialized methods that we briefly discuss next.

In the case where $q \equiv 3 \pmod 4$, one can simply use a specialized version of the Tonelli-Shanks procedure, namely, the Shanks algorithm, where a square root of a QR $a \in \mathbb{F}_q$ is computed via one single exponentiation $b = a^{\frac{q+1}{4}}$. On the other hand, no simple and general formula for the class $q \equiv 1 \pmod 4$ is known. However, fast algorithms for computing square roots in $\mathbb{F}_q$ when $q \equiv 5 \pmod 8$ or $q \equiv 9 \pmod{16}$ have been reported.

For the case $q \equiv 5 \pmod 8$, Atkin [12] developed in 1992 an efficient and deterministic square root algorithm which finds a square root of a QR using only one field exponentiation plus a few multiplications in $\mathbb{F}_q$. A modification of Atkin's algorithm, presented by Müller in [106], computes a square root in $\mathbb{F}_q$ when $q \equiv 9 \pmod{16}$ at the price of two exponentiations. By exploiting a regular structure of the exponents in Müller's algorithm when written in base

[1] In this chapter, an algorithm is said to be non-deterministic if for a given input, the number of steps to compute the output varies among different runs.

p, Kong et al. [89] reduced the overall cost of this procedure to only one exponentiation for half of the QRs in $\mathbb{F}_q$, and two exponentiations for the other half.

It is worth mentioning that in the case where $q \equiv 1 \pmod{16}$, no specialized algorithm is known. Hence, for this class of finite fields one is forced to resort to the aforementioned expensive classical methods, namely, the Tonelli-Shanks algorithm or the improved version of the Cipolla-Lehmer algorithm presented by Müller in [106].

Square root computation in extension fields $\mathbb{F}_{p^m}$, with m odd. Several authors have analyzed the square root problem in odd-degree extension fields. In [18], Barreto et al. presented an efficient algorithm that computes square roots in fields of this form whenever $p \equiv 3 \pmod{4}$ or $p \equiv 5 \pmod{8}$. The latter case can be seen as a variant of the Atkin method mentioned above. The main idea of the Barreto et al. method is to rewrite the resulting exponents in base p. Thus, the exponentiations can be performed efficiently by exploiting a recursive procedure that is essentially the same as the one used in the Itoh-Tsujii inversion method [73]. This recursive procedure takes advantage of the fact that the prime Frobenius map in characteristic p, that is, the p-th powering, is a simple operation that can be computed at an inexpensive cost or even at no cost if the field elements are represented in normal basis [21].

The technique in [18] was systematically applied by Han et al. in [66] for all the specialized methods where $p \equiv 3 \pmod{4}$, $p \equiv 5 \pmod{8}$ or $p \equiv 9 \pmod{16}$. Han et al. also improved the Tonelli-Shanks method that is one of the best choices for tackling the case where $p \equiv 1 \pmod{16}$. Let's write $p^m - 1 = 2^s \cdot t$ with s is a positive integer and t an odd number. Then, in order to compute a square root of an arbitrary QR $a \in \mathbb{F}_q$, the most expensive operation in the Tonelli-Shanks algorithm is the exponentiation $a^{\frac{t-1}{2}}$. As shown in [66], this operation can be considerably sped up by once again exploiting the idea of rewriting the exponent $(t-1)/2$ in base p.

Square root computation in extension fields $\mathbb{F}_{p^m}$, with m even. Relatively less work has been reported for even-degree extension fields. Finding square roots in these fields can sometimes be achieved by *descending* some of the required computations in $\mathbb{F}_{p^m}$ to its proper subfields. In this context, the authors in [84, 85, 121] used a Tonelli-Shanks based approach in order to have most of the computations in subfields of $\mathbb{F}_{p^m}$. More recently, Doliskani and Schost [44] presented an algorithm that takes roots over $\mathbb{F}_{p^m}$ by descending the computation down to the base field $\mathbb{F}_p$ by using the trace function.

Scott adapted in [113] the complex square root formula presented in [50] to the computation of square roots in quadratic extension fields $\mathbb{F}_{q^2}$. The computational cost of his algorithm is that of two square roots, one quadratic residuosity test and one field inversion, all these operations performed over $\mathbb{F}_q$. As will be seen in §6.5, the *complex method* ranks among the most efficient algorithms for computing square roots in even-degree extension fields.

Contributions. The main purpose of this chapter, in joint work with F. Rodríguez-Henríquez [6], is the introduction of two new algorithms for computing square roots in $\mathbb{F}_{q^2}$ with q a power of an odd prime. The two algorithms are complementary in the sense that they cover separately the two congruence classes of odd prime numbers, namely, $q \equiv 1 \pmod{4}$

and $q \equiv 3 \pmod 4$.

For the class $q \equiv 3 \pmod 4$, we present a deterministic procedure that in some sense can be seen as a generalized Shanks algorithm for finite fields with even-degree extension. In this case the proposed algorithm computes a square root in $\mathbb{F}_{q^2}$ by performing two exponentiations, each of them with associated exponent of bitlength $\log_2(q)$.

For the class $q \equiv 1 \pmod 4$, our second algorithm computes a square root in $\mathbb{F}_{q^2}$ by first performing one exponentiation in $\mathbb{F}_{q^2}$ with an exponent of bitlength roughly $\log_2(q)$, followed by the computation of one square root in the subfield $\mathbb{F}_q$.

Our experiments show that the two proposed algorithms are competitive when compared against the complex method, the Tonelli-Shanks and the Müller's procedures. Figure 6.1 presents an overview of the most efficient algorithms for computing square roots in $\mathbb{F}_{p^m}$, when p is an odd prime.

In addition, we present a procedure that computes $\chi_q(a)$, with $q = p^m$ at the cost of several Frobenius exponentiations over $\mathbb{F}_q$ plus the computation of the Legendre symbol in the base field $\mathbb{F}_p$. This procedure is faster than the recursive algorithm proposed by Bach and Huber in [13].

Furthermore, a review of the classical square root algorithms over finite extension fields $\mathbb{F}_q$ is provided. In the case of extension fields $\mathbb{F}_{p^m}$ with m odd, we revisit efficient formulations of several square root algorithms where the quadratic residuosity test of the input is interleaved in such a manner that only some constant number of multiplications is added to the overall computational cost.[2] A detailed complexity analysis of all the reviewed algorithms is also given. In particular and to the best of our knowledge, the complexity analysis of Algorithm 8 that corresponds to the Müller procedure for the subclass $q \equiv 1 \pmod{16}$, has not been reported before in the open literature.

The reminder of the chapter is organized as follows. In Section 6.2 we give some notation and practical settings needed for complexity estimations. In Section 6.3 we present an efficient strategy for performing the quadratic residuosity tests in extension fields. Section 6.4 provides a comprehensive review of known algorithms over extension fields $\mathbb{F}_{p^m}$ with m odd. In Section 6.5, we study the computation of square roots in extension fields $\mathbb{F}_{p^m}$ with m even and outline the new algorithms. In Section 6.6, we compare the new algorithms against previously known methods by choosing base fields of BN curves [20] and NIST curves [72]. Our concluding remarks are drawn in Section 6.7.

6.2 Preliminaries

In this chapter, most of the described algorithms have both *precomputation* and *computation* phases. However, as it is customary when evaluating the complexity of a given algorithm, we will not consider the precomputation effort and give only the costs associated to the computation phase.

In a finite field $\mathbb{F}_q$, the binary exponentiation method is a standard strategy for computing

[2] With the only exception of Algorithm 8 that reproduces one of the procedures that Müller introduced in [106].

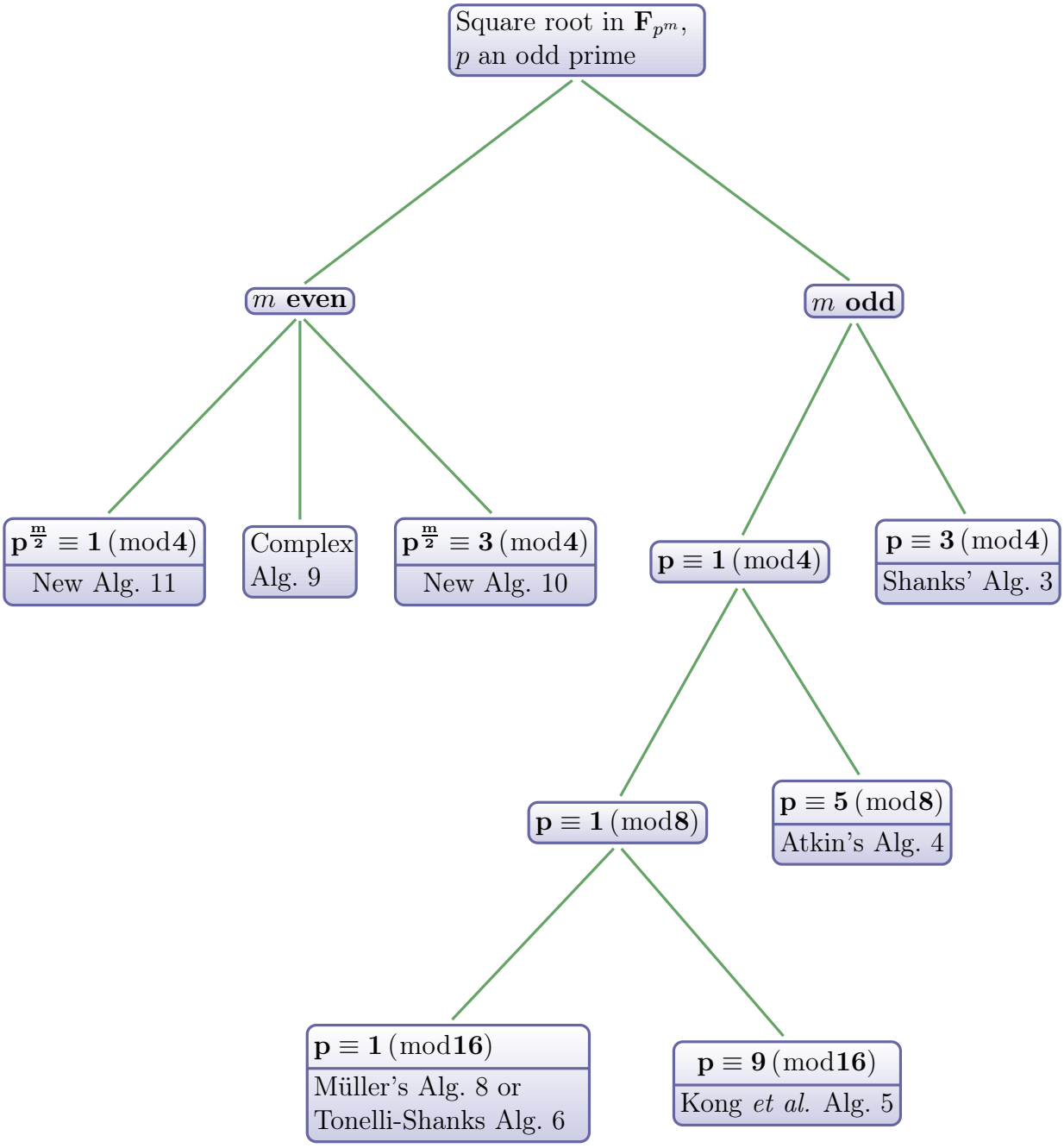

Figure 6.1: A taxonomy of most efficient algorithms for computing square roots in $\mathbb{F}_{p^m}$, where p is an odd prime and $m \geq 1$ an integer.

field exponentiations of the form a^s, with $a \in \mathbb{F}_q$ and $s \leq q-1$ a positive integer. On average, the binary exponentiation requires a total of $\lfloor \log_2(s) \rfloor$ squarings and $\mathrm{Hw}(s)-1$ multiplications in $\mathbb{F}_q$, where $\mathrm{Hw}(s)$ denotes the Hamming weight of s. Throughout this chapter, it will be assumed that the average Hamming weight of a random odd integer s, as given in [97], is $\frac{1}{2}\lfloor \log_2(s) \rfloor + \frac{3}{2}$.

For a quadratic non-residue (QNR) element $\beta \in \mathbb{F}_q$, the binomial polynomial $f(Y) = Y^2 - \beta$ is irreducible in $\mathbb{F}_q[Y]$, which means that the quadratic extension $\mathbb{F}_{q^2}$ of $\mathbb{F}_q$ can be ssen as $\mathbb{F}_q[Y]/(f(Y))$. Let y be a root of $f(Y)$, then an element $a \in \mathbb{F}_{q^2}$ can be represented as $a = a_0 + a_1 y$, for a unique pair $(a_0, a_1) \in \mathbb{F}_q^2$. Therefore, a multiplication in $\mathbb{F}_{q^2}$ can be computed at a cost of three multiplications in $\mathbb{F}_q$ and one multiplication by a constant in $\mathbb{F}_q$, whereas a squaring costs two multiplications in $\mathbb{F}_q$ and two multiplications by a constant in $\mathbb{F}_q$.[3] A multiplication of an element of $\mathbb{F}_{q^2}$ by an element of $\mathbb{F}_q$ costs two multiplications in $\mathbb{F}_q$. We will consider that the addition operation in $\mathbb{F}_{q^2}$ has a negligible cost, and thus will be ignored in our estimations. Let $a = a_0 + a_1 y \in \mathbb{F}_{q^2}$. Since $a^{-1} = (a_0 + a_1 y)^{-1} = (a_0 - a_1 y)/(a_0^2 + \beta \cdot a_1^2)$, computing the inverse of a requires one inversion in $\mathbb{F}_q$ and at most 5 multiplications in $\mathbb{F}_q$ (in fact, if $\beta = -1$ only 4 multiplications in $\mathbb{F}_q$ are required). Applying the q-th Frobenius map to a is essentially free of cost since $a^q = (a_0 + a_1 y)^q = (a_0 - a_1 y)$; a^q is called the *conjugate* of a and denoted by $\bar{a}$. Notice that this also implies that the element $a^{q+1} = a \cdot \bar{a} = a_0^2 - \beta \cdot a_1^2$ lives in $\mathbb{F}_q$. Moreover, if the element a is a QR in $\mathbb{F}_{q^2}$, then $a^{\frac{q+1}{2}}$ also lives in $\mathbb{F}_q$. The application of the Frobenius map over a field element $a \in \mathbb{F}_{q^k}$, with $k > 2$, can be computed efficiently for reasonable choices of irreducible polynomials involved in the construction of the associated *field towering* [23, 87]. In this scenario, the computation of a^q can be achieved at the price of at most $k-1$ field multiplications in $\mathbb{F}_q$ [26].[4]

In the remainder of this chapter, M_q, S_q and Mc_q will denote the cost of a multiplication, a squaring and a multiplication by a constant in $\mathbb{F}_q$, respectively. The cost of an inversion in $\mathbb{F}_q$ is denoted by I_q. We write F_q for the cost of a Frobenius operation a^{p^i}, with $a \in \mathbb{F}_q$, $q = p^m$ and $1 \leq i < m$. $Lucas(k)$ will denote the complexity of computing the k-th term of a Lucas sequence. Finally, we denote by $SQRT_q$ the complexity of computing a square root in $\mathbb{F}_q$ by using the most suitable method for that extension field.

6.3 Reviewing the quadratic residuosity test

Let $a \in \mathbb{F}_q^*$, with $q = p^m$, p an odd prime and $m > 1$. In [13], Bach and Huber showed that the Legendre symbol can be used to compute the quadratic character $\chi_q(a)$. By recursively invoking the law of quadratic reciprocity, Bach proved that the asymptotic cost of its method is $O(\log q)^2$ bit operations. Here, we present an alternative formulation that computes the quadratic character $\chi_q(a)$ by *descending* its computation to the prime field $\mathbb{F}_p$, that is, $\chi_p(a)$, in addition to several Frobenius map applications. This procedure is considerably more efficient than Bach's algorithm provided that the Frobenius map can be applied inexpensively.

[3] using a multiplication *à la* Karatsuba and the so-called complex method, respectively [66, 34].

[4] Note that if the normal basis representation is used then the computation of the Frobenius operator is free.

As already mentioned, the quadratic residuosity test on a can be performed via the exponentiation $a^{\frac{q-1}{2}}$. The following expression of the exponent

$$\frac{q-1}{2} = \frac{p-1}{2}\sum_{i=0}^{m-1} p^i, \tag{6.1}$$

can be used to reduce the exponentiation $a^{\frac{q-1}{2}}$ to one quadratic residuosity test in the base field $\mathbb{F}_p$ after applying the addition chain exponentiation method described in [21]. Indeed, the value $b = a^{\sum_{i=0}^{m-1} p^i}$ is nothing more than the norm of a in $\mathbb{F}_p$, which implies that $b \in \mathbb{F}_p$. For efficiency, notice that after computing b, instead of performing the exponentiation $b^{\frac{p-1}{2}}$, the Legendre symbol computation on $b \in \mathbb{F}_p$ can be carried out. Algorithm 1 implements the above ideas for performing the quadratic residuosity test, where the function $C_{q,k} : \mathbb{F}_q \to \mathbb{F}_s$ is defined as $C_{q,k}(\alpha) = \alpha^{1+s+s^2+\cdots+s^{k-1}}$, for integers k and s such that $s^k = q$.

Algorithm 1 Quadratic residuosity test for $a \in \mathbb{F}_q, q = p^m, m > 1$

Require: $a \in \mathbb{F}_q$, $q = p^m$, $m > 1$.
Ensure: $\chi_q(a)$.
1: $b \leftarrow C_{q,m}(a)$.
2: $c \leftarrow \chi_p(b)$.
3: **return** c.

Let $k|m$ and consider s such that $s^k = q$. For $\alpha \in \mathbb{F}_q$, it can be deduced from [21] that

$$C_{q,k}(\alpha) = \begin{cases} C_{q,n}(\alpha)\Big(C_{q,n}(\alpha)\Big)^{s^n} & \text{if } k = 2n, \\ \Big(C_{q,n}(\alpha)\left(C_{q,n}(\alpha)\right)^{s^n}\Big)^{s}\,\alpha & \text{if } k = 2n+1. \end{cases} \tag{6.2}$$

Algorithm 2 computes $C_{q,k}(\alpha)$ by applying equation (6.2) recursively. It is easy to see that the computational cost of Algorithm 2, out of the recursive call, is either one multiplication and one Frobenius application over $\mathbb{F}_q$, if k is even; or two multiplications and two Frobenius applications over $\mathbb{F}_q$, if k is odd. For the recursivity, Algorithm 2 calls itself exactly $\lfloor \log_2 k \rfloor$ times. Assuming that half of these calls correspond to n even and the other half to n odd, then the overall average complexity for computing $C_{q,k}(\alpha)$ can be estimated as

$$\frac{3}{2}\Big(\lfloor \log_2 k \rfloor + 1\Big)(M_q + F_q).$$

Algorithm 2 Computing $C_{q,k}(\alpha) = \alpha^{1+s+s^2+\cdots+s^{k-1}}$, with $s^k = q$

Require: $\alpha \in \mathbb{F}_q$, $q = p^m$, $k|m$ and $s^k = q$.
Ensure: $C_{k,s}(\alpha) = \alpha^{1+s+s^2+\cdots+s^{k-1}}$
1: **if** $k = 0$ **then**
2: **return** a.
3: **end if**
4: **if** $k \equiv 1 \pmod 2$ **then**
5: $n \leftarrow (k-1)/2$.
6: $C \leftarrow C_{q,n}(\alpha)$
7: $C \leftarrow C \cdot C^{s^{n+1}}$
8: **else**
9: $n \leftarrow k/2$.
10: $C \leftarrow C_{q,n-1}(\alpha)$
11: $C \leftarrow \left(C \cdot C^{s^n}\right)^s \alpha$
12: **end if**
13: **return** C.

Hence, the cost of computing $b = a^{\sum_{i=0}^{m-1} p^i} = C_{q,m}(a)$ is

$$\frac{3}{2}\Big(\lfloor \log_2 m \rfloor + 1\Big)(M_q + F_q).$$

The computation of the Legendre symbol of b has a complexity similar to that of computing the greatest common divisor of b and p [13].

6.4 Square roots in odd-degree extension fields

The algorithms for computing square roots in finite fields $\mathbb{F}_q$, where $q = p^m$ with p an odd prime and m an odd integer, can be classified into two classes. On the one hand, we have the class $q \equiv 1 \pmod 4$, and on the other hand the class $q \equiv 3 \pmod 4$. We first describe the case $q \equiv 3 \pmod 4$ before handling the case $q \equiv 1 \pmod 4$ which can be more expensive as discussed in this section.

6.4.1 Square roots in $\mathbb{F}_q$ when $q \equiv 3 \pmod 4$

Computing a square root of an arbitrary QR $a \in \mathbb{F}_q$ when $q \equiv 3 \pmod 4$ can be done by computing $a^{\frac{q+1}{4}}$. This can be seen as the simplest instance of Shanks's method [114]. The quadratic residuosity test of an input element has been integrated into Algorithm 3. If the element is a QR Algorithm 3 returns a square root of the element and 'false' otherwise.

Algorithm 3 Shanks's algorithm for $q \equiv 3 \pmod 4$

Require: $a \in \mathbb{F}_q^*$.
Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.
1: $a_1 \leftarrow a^{\frac{q-3}{4}}$.
2: $a_0 \leftarrow a_1(a_1 a)$.
3: **if** $a_0 = -1$ **then**
4: **return** false.
5: **end if**
6: $x \leftarrow a_1 a$.
7: **return** x.

The computational cost of Algorithm 3 is that of one exponentiation and two multiplications. In 2007, Scott [113] showed that the complexity of the exponentiation in step 1 of Algorithm 3 could be further reduced by rewriting the exponent in base p. This was rediscovered by Han et al. [66], who expressed the exponent $(q-3)/4$ as

$$\frac{q-3}{4} = \alpha + p\left[p\alpha + (3\alpha+2)\right] \sum_{i=0}^{(m-3)/2} p^{2i}, \tag{6.3}$$

where $\alpha = \frac{p-3}{4}$. Using the expression in equation (6.3), it can be shown that the average complexity of Algorithm 3 when the input is a QR, is given as

$$\left(\frac{1}{2}\lfloor\log_2(p)\rfloor + \frac{3}{2}\lfloor\log_2(m)\rfloor + \frac{5}{2}\right) M_q + \Big(\lfloor\log_2(p)\rfloor - 2\Big) S_q + \left(\frac{3}{2}\lfloor\log_2(m)\rfloor + 2\right) F_q.$$

6.4.2 Square roots in $\mathbb{F}_q$ when $q \equiv 1 \pmod 4$

For this class, it is customary to consider the sub-congruence classes modulo 8 or modulo 16. Indeed, despite the fact that there is no simple and general algorithm for $q \equiv 1 \pmod 4$, fast algorithms for computing square roots in $\mathbb{F}_q$ when $q \equiv 5 \pmod 8$ or $q \equiv 9 \pmod{16}$ are known.

6.4.2.1 Atkin's algorithm

When $q \equiv 5 \pmod 8$, Atkin [12] developed an efficient method for computing square roots in $\mathbb{F}_q$ with one exponentiation and a constant number of multiplications.

The computational cost of Algorithm 4 is that of one exponentiation, four multiplications and two squarings in $\mathbb{F}_q$. Han et al. [66] show that the exponent $(q-5)/8$ can be written in base p as

$$\frac{q-5}{8} = \alpha + p\left[p\alpha + (5\alpha+3)\right] \sum_{i=0}^{(m-3)/2} p^{2i}, \tag{6.4}$$

where $\alpha = \frac{p-5}{8}$. Using the expression in equation (6.4), it can be shown that the average complexity of Algorithm 4 when the input is a QR, is given as

$$\left(\frac{1}{2}\lfloor\log_2(p)\rfloor + \frac{3}{2}\lfloor\log_2(m)\rfloor + 3\right) M_q + \lfloor\log_2(p)\rfloor S_q + \left(\frac{3}{2}\lfloor\log_2(m)\rfloor + 2\right) F_q.$$

Algorithm 4 Atkin's algorithm for $q \equiv 5 \pmod 8$

Require: $a \in \mathbb{F}_q^*$.
Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.

PRECOMPUTATION

1: $t \leftarrow 2^{\frac{q-5}{8}}$.

COMPUTATION

1: $a_1 \leftarrow a^{\frac{q-5}{8}}$.
2: $a_0 \leftarrow (a_1^2 a)^2$.
3: **if** $a_0 = -1$ **then**
4: **return** false.
5: **end if**
6: $b \leftarrow ta_1$.
7: $i \leftarrow 2(ab)b$.
8: $x \leftarrow (ab)(i-1)$.
9: **return** x.

6.4.2.2 Generalized Atkin's algorithm

Atkin's method was generalized first by Müller [106] for the case $q \equiv 9 \pmod{16}$. Müller showed that in this setting, the square root computation for a QR can be achieved at a cost of two exponentiations in $\mathbb{F}_q$. Later, Kong et al. [89] improved this by presenting a procedure that required only one exponentiation for half of the QRs in $\mathbb{F}_q$, and two exponentiations for the remainder. Nonetheless, by precomputing some values, we observed that one can compute a square root at the cost of only one exponentiation as shown in Algorithm 5.

Algorithm 5 Kong et al.'s algorithm for $q \equiv 9 \pmod{16}$

Require: $a \in \mathbb{F}_q^*$.
Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.
PRECOMPUTATION
$c_0 \leftarrow 1$
while $c_0 = 1$ **do**
  Select randomly $c \in \mathbb{F}_q^*$.
  $c_0 \leftarrow \chi_q(c)$.
end while
$d \leftarrow c^{\frac{q-9}{8}}$,
$e \leftarrow c^2$, $t \leftarrow 2^{\frac{q-9}{16}}$.
COMPUTATION
$a_1 \leftarrow a^{\frac{q-9}{16}}$.
$a_0 \leftarrow (a_1^2 a)^4$.
if $a_0 = -1$ **then**
  return false.
end if
$b \leftarrow ta_1$.
$i \leftarrow 2(ab)b$.
$r \leftarrow i^2$.
if $r = -1$ **then**
  $x \leftarrow (ab)(i-1)$.
else
  $u \leftarrow bd$.
  $i \leftarrow 2u^2ea$.
  $x \leftarrow uca(i-1)$.
end if
return x.

The computational complexity of Algorithm 5 is that of one exponentiation, six and a half multiplications, and four and a half squarings in $\mathbb{F}_q$.

For this case, the exponent $(q-9)/16$ can be rewritten in base p as

$$\frac{q-9}{16} = \alpha + p\left[p\alpha + (9\alpha + 5)\right] \sum_{i=0}^{(m-3)/2} p^{2i}, \tag{6.5}$$

where $\alpha = \frac{p-9}{16}$. Using the expression equation (6.5), it can be shown that the average complexity of Algorithm 5 when a is a QR, is given as

$$\left(\frac{1}{2}\lfloor \log_2(p) \rfloor + \frac{3}{2}\lfloor \log_2(m) \rfloor + 10\right) M_q + \left(\lfloor \log_2(p) \rfloor + \frac{5}{2}\right) S_q + \left(\frac{3}{2}\lfloor \log_2(m) \rfloor + 2\right) F_q.$$

6.4.2.3 General square root algorithms in $\mathbb{F}_q$ for $q \equiv 1 \pmod{16}$

This sub-case is certainly the most costly since there is no specialized algorithm to tackle it. The Tonelli-Shanks [114, 120] and the Cipolla-Lehmer [38] algorithms are the two general non-deterministic algorithms from which most of the methods for square root extraction are derived. In this subsection the Tonelli-Shank's algorithm and an improved Cipolla-Lehmer

algorithm by Müller [106] are described. For the latter, we include a detailed analysis of its computational complexity that to the best of our knowledge has not been reported before in the open literature.

Algorithm 6 Tonelli-Shanks Algorithm

Require: $a \in \mathbb{F}_q^*$
Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.

PRECOMPUTATION

1: Write $q - 1 = 2^s t$, where t is odd.
2: $c_0 \leftarrow 1$.
3: **while** $c_0 = 1$ **do**
4: Select randomly $c \in \mathbb{F}_q^*$.
5: $z \leftarrow c^t$.
6: $c_0 \leftarrow c^{2^{s-1}}$.
7: **end while**

COMPUTATION

1: $\omega \leftarrow a^{\frac{t-1}{2}}$.
2: $a_0 \leftarrow (\omega^2 a)^{2^{s-1}}$.
3: **if** $a_0 = -1$ **then**
4: **return** false.
5: **end if**
6: $v \leftarrow s$, $x \leftarrow a\omega$, $b \leftarrow x\omega$.
7: **while** $b \neq 1$ **do**
8: Find least integer $k \geq 0$ such that $b^{2^k} = 1$.
9: $\omega \leftarrow z^{2^{v-k-1}}$, $z \leftarrow \omega^2$, $b \leftarrow bz$, $x \leftarrow x\omega$, $v \leftarrow k$.
10: **end while**
11: **return** x.

Algorithm 6 presents a variant of the Tonelli-Shanks procedure where the quadratic residuosity test on the input has been incorporated. Note that the computational complexity of Algorithm 6 varies depending on whether the input is or is not a QR in $\mathbb{F}_q$. By taking into account the average contribution of QR and QNR inputs, and using the complexity analysis given in [97] for the classical Tonelli-Shanks algorithm, it is not difficult to see that the average computational cost of Algorithm 6 is given as

$$\frac{1}{2}\Big(\lfloor \log_2(q) \rfloor + 4\Big) M_q + \left(\lfloor \log_2(q) \rfloor + \frac{1}{8}\big(s^2 + 3s - 16\big) + \frac{1}{2^s}\right) S_q. \tag{6.6}$$

However, rewriting the exponent $(t-1)/2$ in base p as

$$\frac{t-1}{2} = \alpha + p\left[\alpha(p+1) + 1 + 2^{s-1}t\right] \sum_{i=0}^{(m-3)/2} p^{2i},$$

where $q - 1 = 2^s t$, $p - 1 = 2^s u$, and $\alpha = \frac{u-1}{2}$, it can be shown that the average complexity of Algorithm 6 for arbitrary field element inputs is given as

$$\left(\frac{1}{2}\lfloor \log_2(p) \rfloor + \frac{3}{2}\lfloor \log_2(m) \rfloor + \frac{s}{2} + 5\right) M_q + \left(\lfloor \log_2(p) \rfloor + \frac{1}{8}(s^2 + 11s - 16) + \frac{1}{2^s}\right) S_q + \left(\frac{3}{2}\lfloor \log_2(m) \rfloor + 2\right) F_q.$$

Algorithm 7 Lucas sequence evaluation

Require: $\alpha \in \mathbb{F}_q$ and $k \geq 2$.
Ensure: $V_k(\alpha, 1)$.
1: Write $k = \sum_{j=0}^{l-1} b_j 2^j$ in binary form.
2: $d_0 \leftarrow \alpha$.
3: $d_1 \leftarrow \alpha^2 - 2$.
4: **for** j from $l-2$ to 1 **do**
5: $d_{1-b_j} \leftarrow d_0 d_1 - \alpha$, $d_{b_j} \leftarrow d_{1-b_j}^2 - 2$.
6: **end for**
7: **if** $b_0 = 1$ **then** $v \leftarrow d_0 d_1 - \alpha$ **else** $v \leftarrow d_0^2 - 2$.
8: **return** v.

As a second option for this sub-case, the improved Cipolla-Lehmer algorithm [106] uses Lucas sequences to compute square roots in $\mathbb{F}_q$. We briefly recall the definition of Lucas sequences and subsequently give a fast algorithm that evaluates the k-th term of some instances of these sequences. For $\alpha, \beta \in \mathbb{F}_q$, the Lucas sequence $\Big(V_k(\alpha, \beta)\Big)_{k \geq 0}$ is defined as

$$V_0 = 2, \quad V_1 = \alpha \text{ and } V_k = \alpha V_{k-1} - \beta V_{k-2}, \text{ for } k > 1.$$

Algorithm 7 computes $V_k(\alpha, 1)$ for given $\alpha \in \mathbb{F}_q$ and $k > 1$. It can be easily verified that to compute $V_k(\alpha, 1)$, this procedure requires roughly $(\lfloor \log_2(k) \rfloor + \frac{3}{2})S_q + (\lfloor \log_2(k) \rfloor + \frac{1}{2})M_q$.

Algorithm 8 Müller's algorithm [106]

Require: $a \in \mathbb{F}_q^*$.
Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.
1: **if** $a = 4$ **then**
2: **return** 2.
3: **end if**
4: $t \leftarrow 1$.
5: $a_1 \leftarrow \chi_q(at^2 - 4)$.
6: **while** $a_1 = 1$ **do**
7: Select randomly $u \in \mathbb{F}_q^* \backslash \{1\}$.
8: $t \leftarrow u$.
9: **if** $at^2 - 4 = 0$ **then**
10: **return** $2t^{-1}$.
11: **end if**
12: $a_1 \leftarrow \chi_q(at^2 - 4)$.
13: **end while**
14: $\alpha \leftarrow at^2 - 2$.
15: $x \leftarrow V_{\frac{q-1}{4}}(\alpha, 1)/t$.
16: $a_0 \leftarrow x^2 - a$.
17: **if** $a_0 \neq 0$ **then**
18: **return** false.
19: **end if**
20: **return** x.

Algorithm 8 essentially describes the same square root algorithm as presented in [106]. In order to assess the computational complexity of this procedure, we present the following two lemmas.

Lemma 6.1. In the field $\mathbb{F}_q$, the number of QR $a \in \mathbb{F}_q^*$ such that $a - 4$ is a QNR is $\frac{q-1}{4}$.

Proof. To prove this, one can first compute the number of QR $a \in \mathbb{F}_q^*$ such that $a-4$ is a QR, which is clearly half of the number of $b \in \mathbb{F}_q^*$ such that $b^2 - 4$ is a QR. It is shown in [119, Lemma 3.1] that $\#\{b \in \mathbb{F}_q \mid b^2 - 4 \text{ is a QR in } \mathbb{F}_q\} = \frac{q+1}{2}$. Now, when $b = 0$, -4 is a QR in $\mathbb{F}_q$, since $q \equiv 1 \pmod 4$. Thus, we have $\#\{b \in \mathbb{F}_q^* \mid b^2 - 4 \text{ is a QR in } \mathbb{F}_q\} = \frac{q-1}{2}$, and then

$\#\{a \in \mathbb{F}_q^* \mid a \text{ and } a-4 \text{ are QRs in } \mathbb{F}_q\} = \frac{q-1}{4}$. Hence, the number of QR $a \in \mathbb{F}_q^*$ such that $a-4$ is a QNR is $\frac{q-1}{2} - \frac{q-1}{4} = \frac{q-1}{4}$. □

Lemma 6.2. Let $a \in \mathbb{F}_q^*$ be a QR. Then the number of $t \in \mathbb{F}_q^*$ such that $at^2 - 4$ is a QNR is $\frac{q-1}{2}$.

Proof. As in the proof of Lemma 6.1, let's start by computing the number of $t \in \mathbb{F}_q^*$ such that $at^2 - 4$ is a QR, that is, the number of $t \in \mathbb{F}_q^*$ such that there exists $s \in \mathbb{F}_q$ with $at^2 - 4 = s^2$. For such a t, $at^2 - 4 = s^2$ is equivalent to $a - 4r^2 = s^2r^2$, where $r = t^{-1}$, and then to $a = (s^2+4)r^2$. Thus the number of these t is equal to the number of $r \in \mathbb{F}_q$ such that there exist $s \in \mathbb{F}_q$ and $a = (s^2+4)r^2$.
Claim: The number of the above r's is double the number of QRs $c \in \mathbb{F}_q^*$ such that $c-4$ is also a QR in $\mathbb{F}_q$.
Indeed, suppose that we have such a c. Let $s = \pm\sqrt{c-4}$, then $s^2 + 4 = c$. Hence, it can be seen that for each such c, one obtains two solutions to the equation $a = (s^2+4)r^2$, namely, $r_{1,2} = \pm\sqrt{a/(s^2+4)}$. Moreover, since for a different c' with properties as for c this procedure gives two elements (r_1', r_2') with $(r_1', r_2') \neq (r_1, r_2)$ and $(r_1', r_2') \neq (r_2, r_1)$, in addition to the fact that the above procedure is reversible, one can see that

$$\#\{r \in \mathbb{F}_q \mid \exists\ s \in \mathbb{F}_q \text{ and } a = (s^2+4)r^2\} = 2\#\{c \in \mathbb{F}_q^* \mid c \text{ and } c-4 \text{ are QRs in } \mathbb{F}_q\}.$$

Recalling from the proof of Lemma 6.1, we have $\#\{c \in \mathbb{F}_q^* \mid c-4 \text{ is a QR in } \mathbb{F}_q\} = \frac{q-1}{4}$, and therefore $\#\{t \in \mathbb{F}_q^* \mid at^2 - 4 \text{ is a QR in } \mathbb{F}_q\} = \frac{q-1}{2}$. Hence, the number of $t \in \mathbb{F}_q^*$ such that $at^2 - 4$ is a QNR is $q - 1 - \frac{q-1}{2} = \frac{q-1}{2}$. □

Summarizing, Lemma 6.1 shows that for half of the QRs in $\mathbb{F}_q^*$, there is no need to search for a t in the main loop of Algorithm 8, and Lemma 6.2 ensures that for the remainder case only 2 iterations in the while-loop suffice on average. Thus, the expected number of multiplications and squarings, in the cases where $(a-4)^{\frac{q-1}{2}} = -1$ and $(a-4)^{\frac{q-1}{2}} = 1$, can be computed. Recall that I_q denotes the complexity of computing an inverse in $\mathbb{F}_q$. We have

- If $(a-4)^{\frac{q-1}{2}} = -1$, on average, one has to compute one exponentiation and one Lucas sequence evaluation.
- If $(a-4)^{\frac{q-1}{2}} = 1$, on average, one has to compute three exponentiations, one Lucas sequence evaluation, one inversion, two multiplications and two squarings.

Once again, notice that the exponentiation of step 5 can be optimized by rewriting the exponent $(q-1)/2$ as

$$\frac{q-1}{2} = \frac{p-1}{2} \sum_{i=0}^{(m-1)} p^i,$$

which gives an expected computational cost of Algorithm 8 in all QRs in $\mathbb{F}_q$ as

$$\left(\lfloor \log_2(q) \rfloor + \frac{15}{4}\lfloor \log_2(m) \rfloor + \frac{13}{4}\right) M_q + \left(\lfloor \log_2(q) \rfloor - \frac{1}{2}\right) S_q + \left(\frac{15}{4}\lfloor \log_2(m) \rfloor + \frac{17}{4}\right) F_q$$
$$+\Big(\lfloor \log_2(p) \rfloor - 3\Big) M_p + \Big(2\lfloor \log_2(p) \rfloor - 2\Big) S_p + \frac{1}{2} I_p.$$

6.5 Square roots in even-degree extension fields

In this setting, none of the methods studied in the previous section leads to efficient computation of square roots as it is briefly discussed in the following. Let q be a power of an odd prime. Then, the congruence $q^2 \equiv 1 \pmod 4$ always holds. Moreover, it is easy to see that the case $q^2 \equiv 5 \pmod 8$ never occurs. This automatically implies that the Shanks's and the Atkin's methods are both ruled out. In the case where $q^2 \equiv 9 \pmod{16}$, one can use the generalized Atkin's algorithm by Kong et al.. If however $q^2 \equiv 1 \pmod{16}$, the only remaining option is classically to select between the Tonelli-Shanks algorithm and the Müller algorithm.

In this section, three efficient methods for computing square roots in even-degree extension fields are discussed. First, a detailed analysis of the complex method described in [113] is given. Subsequently, we present two novel algorithms for computing square roots in $\mathbb{F}_{q^2}$. The two new algorithms are complementary in the sense that they cover separately the two congruence classes of odd primes, namely, $q \equiv 1 \pmod 4$ and $q \equiv 3 \pmod 4$. The easiest case $q \equiv 3 \pmod 4$ is first outlined, followed by the slightly more involved case $q \equiv 1 \pmod 4$.

Algorithm 9 Complex method for square root computation in $\mathbb{F}_{q^2}$

Require: Irreducible binomial $f(y) = y^2 - \beta$ such that $\mathbb{F}_{q^2} \cong \mathbb{F}_q[y]/\left(y^2 - \beta\right)$, $\beta \in \mathbb{F}_q$, with $q = p^n$, $a = a_0 + a_1 y \in \mathbb{F}_{q^2}^*$.
Ensure: If it exists, $x = x_0 + x_1 y \in \mathbb{F}_{q^2}$ satisfying $x^2 = a$, false otherwise.
1: **if** $a_1 = 0$ **then**
2: **return** $\mathrm{SQRT}_q(a_0)$.
3: **end if**
4: $\alpha \leftarrow a_0^2 - \beta \cdot a_1^2$.
5: $\gamma \leftarrow \chi_q(\alpha)$.
6: **if** $\gamma = -1$ **then**
7: **return** false.
8: **end if**
9: $\alpha \leftarrow \mathrm{SQRT}_q(\alpha)$.
10: $\delta \leftarrow \frac{a_0+\alpha}{2}$.
11: $\gamma \leftarrow \chi_q(\delta)$.
12: **if** $\gamma = -1$ **then**
13: $\delta \leftarrow \frac{a_0-\alpha}{2}$.
14: **end if**
15: $x_0 \leftarrow \mathrm{SQRT}_q(\delta)$.
16: $x_1 \leftarrow \frac{a_1}{2x_0}$.
17: $x \leftarrow x_0 + x_1 y$.
18: **return** x.

6.5.1 The complex method

Let $q = p^n$, with p an odd prime and $n \geq 1$. Define the quadratic extension field $\mathbb{F}_{q^2}$ as $\mathbb{F}_{q^2} \cong \mathbb{F}_q[y]/\left(y^2 - \beta\right)$, where β is a QNR in $\mathbb{F}_q$. Let $a = a_0 + a_1 y$ be an arbitrary QR in $\mathbb{F}_{q^2}^*$. For a square root $x = x_0 + x_1 y \in \mathbb{F}_{q^2}$ of a, since $x^2 = {x_0}^2 + 2x_0x_1y + \beta{x_1}^2$, x_0 and x_1 must satisfy the following two equations

$$\begin{cases} {x_0}^2 + \beta{x_1}^2 = a_0, \\ 2x_0x_1 = a_1. \end{cases}$$

Solving this system in x_0 and x_1 yields

$$\begin{aligned} x_0 &= \left(\frac{a_0 \pm ({a_0}^2 - \beta {a_1}^2)^{\frac{1}{2}}}{2} \right)^{\frac{1}{2}} \\ x_1 &= \frac{a_1}{2x_0}. \end{aligned} \tag{6.7}$$

Note that ${a_0}^2 - \beta {a_1}^2$ is a QR in $\mathbb{F}_q$ if and only if a is a QR in $\mathbb{F}_{q^2}$, as can be easily checked:

$$\begin{aligned} (a_0 + a_1 y)^{\frac{q^2-1}{2}} &= \left((a_0 + a_1 y)^{q+1}\right)^{\frac{q-1}{2}} \\ &= \left((a_0 - a_1 y) \cdot (a_0 + a_1 y)\right)^{\frac{q-1}{2}} \\ &= \left({a_0}^2 - \beta {a_1}^2\right)^{\frac{q-1}{2}}. \end{aligned}$$

Algorithm 9 implements the complex method for computing square roots in the quadratic extension $\mathbb{F}_{q^2}$ by performing two quadratic residuosity tests (in steps 5 and 11), which can be computed efficiently by using the method described in §6.3. Besides these two tests, the cost of Algorithm 9 also includes that of computing two square roots in $\mathbb{F}_q$ in addition to one field inversion in $\mathbb{F}_q$.

6.5.2 A deterministic algorithm when $q \equiv 3 \pmod 4$

A technique for computing a square root of a QR $a \in \mathbb{F}_{q^2}$ is to find an element $b \in \mathbb{F}_{q^2}$ for which there exists an odd integer s such that $b^2 a^s = 1$. In this case, a square root of a is given by $ba^{\frac{s+1}{2}}$. In order to take advantage of the above property, we proceed as follows.

Let b and s be defined as $b = (1 + a^{\frac{q-1}{2}})^{\frac{q-1}{2}}$ and $s = \frac{q-1}{2}$. Let's first consider the case where $b \neq 0$. Then, the identity $b^2 a^s = 1$ holds since:

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

Conversely, if $b = 0$, from the definition of b we have $1 + a^{\frac{q-1}{2}} = 0$, whence $a^{\frac{q-1}{2}} = -1$. Therefore, $x = ia^{\frac{q+1}{4}}$ is a square root of a, where $i = \sqrt{-1}$, since $x^2 = i^2 a^{\frac{q+1}{2}} = i^2 a^{\frac{q-1}{2}} a = (-1)(-1)a = a$.

In practice, the value of i can be readily found whenever the quadratic extension field $\mathbb{F}_{q^2}$ is constructed using a binomial $f(Y) = Y^2 - \beta$, with β a QNR in $\mathbb{F}_q$. In this case, if y is a root of $f(Y)$, then $i = \beta^{\frac{q-3}{4}} y$ yields $i^2 = \beta^{\frac{q-3}{2}} y^2 = \beta^{\frac{q-3}{2}} \beta = \beta^{\frac{q-1}{2}} = -1$ as required. However, since $q \equiv 3 \pmod 4$, one can typically choose $\beta = -1$ and therefore set $i = y$.

Summarizing, a square root x of a QR $a \in \mathbb{F}_{q^2}$, with $q \equiv 3 \pmod 4$ can be found with

$$x = \begin{cases} i a^{\frac{q+1}{4}} & \text{if } a^{\frac{q-1}{2}} = -1, \\ \left(1 + a^{\frac{q-1}{2}}\right)^{\frac{q-1}{2}} a^{\frac{q+1}{4}} & \text{otherwise.} \end{cases} \tag{6.8}$$

Notice the striking similarity between the classic Shanks's algorithm in §6.4 and this method. Thus one can view equation (6.8) as a generalization of Shanks's method for the considered even-degree extension fields.

Algorithm 10 Square root computation in $\mathbb{F}_{q^2}$ when $q \equiv 3 \pmod 4$

Require: $a \in \mathbb{F}_{q^2}^*$, $i \in \mathbb{F}_{q^2}$, such that $i = \sqrt{-1}$, with $q = p^n$.
Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.
1: $a_1 \leftarrow a^{\frac{q-3}{4}}$.
2: $\alpha \leftarrow a_1(a_1 a)$.
3: $a_0 \leftarrow \alpha^q \alpha$.
4: **if** $a_0 = -1$ **then**
5: **return** false.
6: **end if**
7: $x_0 \leftarrow a_1 a$.
8: **if** $\alpha = -1$ **then**
9: $x \leftarrow i x_0$.
10: **else**
11: $b \leftarrow (1 + \alpha)^{\frac{q-1}{2}}$.
12: $x \leftarrow b x_0$.
13: **end if**
14: **return** x.

Algorithm 10 implements a procedure for computing square roots using equation (6.8). After executing steps 1-3, the variables α and a_0 are assigned as $\alpha = a^{(q-1)/2}$ and $a_0 = a^{(q^2-1)/2}$. Therefore, in steps 4-6 the quadratic residuosity test on a is performed in $\mathbb{F}_{q^2}$. In the case where a is not a QR, the algorithm returns 'false'. Otherwise, after executing step 7 the variable x_0 is assigned as $x_0 = a^{(q+1)/4}$. Then, according to equation (6.8), if in step 8 it is determined that $\alpha = -1$, a square root of a is returned as $x = i x_0$. Otherwise, in step 11, b is computed as $b = \left(1 + a^{\frac{q-1}{2}}\right)^{\frac{q-1}{2}}$ and the value of a square root of a is computed in step 12 as $x = b x_0$.

Algorithm 10 performs at most two exponentiations in $\mathbb{F}_{q^2}$, in steps 1 and 11. Additionally, in steps 2, 3, 7 and 12 a total of five multiplications in $\mathbb{F}_{q^2}$ are required. As already seen in §6.4, the exponent $(q-3)/4$ in step 1 can be written in terms of p as

$$\frac{q-3}{4} = \alpha + p\left[p\alpha + (3\alpha + 2)\right] \sum_{i=0}^{(n-3)/2} p^{2i},$$

where $\alpha = \frac{p-3}{4}$. Similarly, the exponent of $(q-1)/2$ in step 11 can be written in base p as

$$\frac{q-1}{2} = \frac{p-1}{2} + \sum_{i=0}^{n-1} p^i.$$

Therefore, the exponentiation $a^{\frac{q-3}{4}}$ can be computed by performing the exponentiation $a^{\frac{p-3}{4}}$, four multiplications in $\mathbb{F}_{q^2}$, one squaring in $\mathbb{F}_{q^2}$ and two Frobenius over $\mathbb{F}_{q^2}$, plus one

evaluation of the sequence $C_{q^2,2}$ that can be computed using Algorithm 2. The average costs of computing $a^{\frac{p-3}{4}}$ and $C_{q^2,2}$ are, respectively,

$$\left(\frac{1}{2}\lfloor\log_2(p)\rfloor - \frac{3}{2}\right) M_{q^2} + \Big(\lfloor\log_2(p)\rfloor - 2\Big) S_{q^2} \quad \text{and} \quad \frac{3}{2}\lfloor\log_2 n\rfloor (M_{q^2} + F_{q^2}).$$

Similarly, the exponentiation $(1+\alpha)^{(q-1)/2}$ can be computed by performing the exponentiation $(1+\alpha)^{(p-1)/2}$, one multiplication and one evaluation of the sequence $C_{q^2,n}$. Hence, the overall average computational cost associated to Algorithm 10 for a QR input in $\mathbb{F}_{q^2} = \mathbb{F}_{p^m}$ is given as

$$\Big(\lfloor\log_2(p)\rfloor + 3\lfloor\log_2 n\rfloor + 7\Big) M_{q^2} + \Big(2\lfloor\log_2(p)\rfloor - 2\Big) S_{q^2} + \Big(3\lfloor\log_2 n\rfloor + 4\Big) F_{q^2}.$$

6.5.3 A descending algorithm when $q \equiv 1 \pmod 4$

The main idea of Algorithm 11 is to descend the square root problem from $\mathbb{F}_{q^2}$ to $\mathbb{F}_q$ via one exponentiation with exponent of bitlength $\log_2(q)$ plus some precomputations. Let $a \in \mathbb{F}_{q^2}$ be a QR of which we want a square root. The approach of descending the square root computation from $\mathbb{F}_{q^2}$ to $\mathbb{F}_q$ can be achieved by exploiting the opportunistic identity

$$\begin{aligned} a &= a\left(a^{\frac{q-1}{2}}\right)^{q+1} \\ &= a\left(a^{\frac{q-1}{2}}\right)^{q} a^{\frac{q-1}{2}} \\ &= \left(a^{\frac{q-1}{2}}\right)^{q} a^{\frac{q+1}{2}}, \end{aligned} \tag{6.9}$$

where the first equality is obtained from the fact that $a^{\frac{q^2-1}{2}} = 1$, since a in a QR in $\mathbb{F}_{q^2}$. Then, by taking square roots in both sides of equation (6.9) we get

$$\sqrt{a} \;=\; \pm\left(a^{\frac{q-1}{4}}\right)^{q} \sqrt{a^{\frac{q+1}{2}}}. \tag{6.10}$$

Now, since $\left(a^{\frac{q+1}{2}}\right)^{q-1} = a^{\frac{q^2-1}{2}} = 1$, the element $a^{\frac{q+1}{2}}$ lives in the subfield $\mathbb{F}_q$. Moreover, if $a^{\frac{q+1}{2}}$ is a QR in $\mathbb{F}_q$, then $\left(a^{\frac{q+1}{2}}\right)^{\frac{q-1}{2}} = 1$ holds. This implies that the problem of finding square roots in $\mathbb{F}_{q^2}$ can be reduced to the problem of finding square roots in its proper subfield $\mathbb{F}_q$ via one exponentiation with an exponent of roughly the same bitlength as q. In the case where $a^{\frac{q+1}{2}}$ is not a QR in $\mathbb{F}_q$, precomputing a QNR in $\mathbb{F}_{q^2}$ allows easily to recover the previous case, as given in Algorithm 11.

Algorithm 11 Square root computation in $\mathbb{F}_{q^2}$ when $q \equiv 1 \pmod 4$

Require: $a \in \mathbb{F}_{q^2}^*$, with $q = p^n$, $n \geq 1$.
Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.
PRECOMPUTATION
1: $c_0 \leftarrow 1$.
2: **while** $c_0 = 1$ **do**
3: Select randomly $c \in \mathbb{F}_{q^2}^*$.
4: $c_0 \leftarrow \chi_{q^2}(c)$.
5: **end while**
6: $d \leftarrow c^{\frac{q-1}{2}}$.
7: $e \leftarrow (dc)^{-1}$.
8: $f \leftarrow (dc)^2$.
COMPUTATION
1: $b \leftarrow a^{\frac{q-1}{4}}$.
2: $a_0 \leftarrow (b^2)^q b^2$.
3: **if** $a_0 = -1$ **then**
4: **return** false.
5: **end if**
6: **if** $b^q b = 1$ **then**
7: $x_0 \leftarrow \mathrm{SQRT}_q(b^2 a)$.
8: $x \leftarrow x_0 b^q$.
9: **else**
10: $x_0 \leftarrow \mathrm{SQRT}_q(b^2 af)$.
11: $x \leftarrow x_0 b^q e$.
12: **end if**
13: **return** x.

Theorem 6.3. Algorithm 11 returns a square root (if it exists) of an input $a \in \mathbb{F}_{q^2}$ via one exponentiation with exponent of bitlength $\log_2(q)$ in $\mathbb{F}_{q^2}$ and one square root computation in the subfield $\mathbb{F}_q$.

Proof. At step 2 of the computation phase, the value of a_0 is,

$$(b^2)^q b^2 = (b^2)^{q+1} = \left[(a^{\frac{q-1}{4}})^2\right]^{q+1} = (a^{\frac{q-1}{2}})^{q+1},$$

which corresponds to the quadratic residuosity test of a in $\mathbb{F}_{q^2}$. Thus, if $a_0 = -1$ a is a QNR in $\mathbb{F}_{q^2}$ and 'false' is returned. We assume that a is a QR ($a_0 = 1$) in what follows.

In step 6, it is tested whether $b^q b = b^{q+1} = (a^{\frac{q+1}{2}})^{\frac{q-1}{2}}$ is 1 or not. If it is 1, then $a^{\frac{q+1}{2}}$ is a QR in $\mathbb{F}_q$. In this case, at step 7, a square root x_0 of $b^2 a = a^{\frac{q+1}{2}}$ in $\mathbb{F}_q$ is computed and a square root of a is then given as $x = x_0 b^q$, since

$$x^2 = x_0^2 b^{2q} = a^{\frac{q+1}{2}} \left(a^{\frac{q-1}{4}}\right)^{2q} = a a^{\frac{q-1}{2}} \left(a^{\frac{q-1}{2}}\right)^q a \left(a^{\frac{q-1}{2}}\right)^{q+1} = a a_0 = a.$$

Now, let's assume that $b^q b = -1$. Note that $d^q d = d^{q+1} = c^{\frac{q^2-1}{2}} = -1$ since c is set as a QNR in $\mathbb{F}_{q^2}$ from the precomputation phase. At step 10, it is easy to see that the element $b^2 af$ lies in $\mathbb{F}_q$ where it is a QR. To see this, consider the following

$$(b^2 af)^{\frac{q-1}{2}} = b^{q-1} a^{\frac{q-1}{2}} (dc)^{q-1} = b^{q-1} b^2 d^{q-1} d^2 (b^q b)(d^q d) = (-1)(-1) = 1.$$

After a square root x_0 of $b^2 af$ in $\mathbb{F}_q$ is computed, it is now easy to see that $x = x_0 b^q e$ is a square root of a since

$$x^2 = (x_0 b^q e)^2 = b^2 af b^{2q} e^2 = a b^{2q+2} (dc)^2 \left[(dc)^{-1}\right]^2 a b^{2q+2} = a a_0 = a.$$

□

The cost of Algorithm 11 includes the computation of one field exponentiation in $\mathbb{F}_{q^2}$, one square root in $\mathbb{F}_q$, 5 field multiplications, one squaring and two Frobenius over $\mathbb{F}_{q^2}$. The exponent $(q-1)/4$ of step 1 can be written in base p as $\frac{q-1}{4} = \frac{p-1}{4} + \sum_{i=0}^{n-1} p^i$. Thus, $a^{(q-1)/4}$ is computed by performing the exponentiation $a^{\frac{p-1}{4}}$, one multiplication and one evaluation of the sequence $C_{q^2}(n,1)$. Hence, the overall average computational cost associated to Algorithm 11 for a QR input in $\mathbb{F}_{q^2} = \mathbb{F}_{p^m}$ is given as

$$\left(\frac{1}{2}\lfloor \log_2(p) \rfloor + \frac{3}{2}\lfloor \log_2 m \rfloor + \frac{11}{2}\right) M_{q^2} + \Big(\lfloor \log_2(p) \rfloor - 2\Big) S_{q^2} + \left(\frac{3}{2}\lfloor \log_2 m \rfloor + 3\right) F_{q^2} + SQRT_q.$$

6.6 Experimental comparisons

In this section, we compare the algorithms described above for the cases where one wants to compute square roots in $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$, with p an odd prime. In our experiments, two group of primes have been considered. The first group is composed of primes $p \equiv 3 \pmod 4$ where Algorithm 10 applies. The second one considers primes $p \equiv 1 \pmod 4$, where Algorithm 11 applies. The extensions $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$ are obtained by constructing the following field towering,

$$\mathbb{F}_p \subset \mathbb{F}_{p^3} \subset \mathbb{F}_{p^6} \subset \mathbb{F}_{p^{12}}.$$

In these experiments, BN primes [20] and NIST recommended primes for elliptic curve cryptography [72] are selected. These choices are made because computing square roots in extensions fields is required in pairing-based cryptography and elliptic curve cryptography. It is worth mentioning that BN curves are a rich family of low embedding-degree elliptic curves defined over a prime field $\mathbb{F}_p$, where p is parametrized as $p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$, with $u \in \mathbb{Z}$. For the sake of simplicity we assume in the following that $M_p = S_p$.

For comparisons over $\mathbb{F}_{p^2}$, the quadratic extension is constructed as $\mathbb{F}_{p^2} = \mathbb{F}_p[U]/(U^2 - \beta)$, where β is a QNR in $\mathbb{F}_p$. Hence, if u is a root of $U^2 - \beta$, all element a in $\mathbb{F}_{p^2}$ can be represented as $a = a_0 + a_1 u$, yielding: $M_{p^2} = 3M_p + 1Mc_p$, $S_{p^2} = 2M_p + 2Mc_p$, $I_{p^2} = I_p + 4M_p + Mc_p$. Analogous costs also hold for the quadratic extensions $\mathbb{F}_{p^3} \subset \mathbb{F}_{p^6}$ and $\mathbb{F}_{p^6} \subset \mathbb{F}_{p^{12}}$.

The cubic extension $\mathbb{F}_p \subset \mathbb{F}_{p^3}$ is obtaining by considering a cubic non-residue $\xi \in \mathbb{F}_p$. We chose $p \equiv 1 \pmod 3$ in order to have a simple way for finding cubic non-residues. Indeed, in this case, an element $\xi \in \mathbb{F}_p$ is a cubic non-residue if and only if $\xi^{\frac{p-1}{3}} \neq 1$. Let $\xi \in \mathbb{F}_p$ be a cubic non-residue, then the cubic field extension $\mathbb{F}_{p^3}$ can be built as $\mathbb{F}_p[U]/(U^3 - \xi)$. Let u be a root of $U^3 - \xi$, then an element α of $\mathbb{F}_{p^3}$ is represented as $\alpha_2 u^2 + \alpha_1 u + \alpha_0 u$, with α_0, α_1 and $\alpha_2 \in \mathbb{F}_p$. The above construction leads to the following arithmetic costs $M_{p^3} = 6M_p + 2Mc_p$, $S_{p^3} = 5M_p + 2Mc_p$, $I_{p^3} = I_p + 12M_p + 4Mc_p$.

Tables 6.1-6.4 present our experimental results in terms of number of general field multiplications, multiplications by a constant, and inversions in $\mathbb{F}_p$, for different choices of odd primes p.[5] For the case $p \equiv 3 \pmod 4$, it can be seen from Tables 6.1 and 6.3 that the

[5]The corresponding Maple and magma scripts can be downloaded at:

complex method is the most efficient procedure followed by Algorithm 10. In the case where $p \equiv 1 \pmod 4$, it can be seen from Tables 6.2 and 6.4 that the complex method and Algorithm 11 are the two most efficient solutions. All these three algorithms are considerably faster than the classical Tonelli-Shanks and Müller's procedures. In the scenario where the multiplication by constants has negligible cost (for example when the irreducible binomial that is used to build the extension field has a constant term of value ± 1), then Algorithm 11 outperforms the complex method.

6.7 Concluding remarks

In this chapter, the computation of square roots in extension fields of the form $\mathbb{F}_{q^2}$, where q is a power of an odd prime, has been studied. We introduced two novel algorithms for the cases $q \equiv 1 \pmod 4$ (Algorithm 10) and $q \equiv 3 \pmod 4$ (Algorithm 11). From our complexity analysis and the experimental comparisons, we conclude that in the case where $q \equiv 3 \pmod 4$, the complex method [113] is the most efficient option, while Algorithm 10, thanks to its simplicity and its deterministic character, offers a good alternative in many cryptographic applications (see [37]). In the case where $q \equiv 1 \pmod 4$, in many practical cases Algorithm 11 is the most efficient approach, closely followed by the complex method.

Table 6.1: Number of operations in $\mathbb{F}_p$ for square roots in $\mathbb{F}_{q^2}$, $q = p$, $p \equiv 3 \pmod 4$

Parameter		$u = -(2^{62} + 2^{55} + 1)$	$p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	$u = 2^{63} + 2^9 + 2^8 + 2^6 + 2^4 + 2^3 + 1$
Bit length of p		254	256	258
s: $p^2 - 1 = 2^s t$, t odd		3	97	4
Algo. 9	M_p	1261	1785	1427
	Mc_p	1091	1271	1157
	I_p	0	0	0
Complex Algo.	M_p	885	1149	972
	Mc_p	6	6	6
	I_p	1	1	1
Tonelli-Shanks	M_p	1574	6292	1660
	Mc_p	1202	6999	1244
	I_p	0	0	0
Müller's Algo.	M_p	3120	3387	3245
	Mc_p	1521	1537	1546
	I_p	1	1	1

`http://delta.cs.cinvestav.mx/~francisco/codigo.html`.

Table 6.2: Number of operations in $\mathbb{F}_p$ for square roots in $\mathbb{F}_{q^2}$, $q = p$, $p \equiv 1 \pmod 4$

Parameter		$p = 2^{224} - 2^{96} + 1$	$u = 2^{62} - 2^{54} + 2^{44}$	$u = 2^{63} - 2^{49}$
Bit length of p		224	254	256
s: $p^2 - 1 = 2^s t$, t odd		97	46	51
Algo. 10	M_p	1975	1625	1782
	Mc_p	577	591	603
	I_p	1	0	0
Complex Algo.	M_p	2653	2079	2357
	Mc_p	7	5	5
	I_p	3	1	1
Tonelli-Shanks	M_p	6705	2934	3199
	Mc_p	6065	2402	2669
	I_p	0	0	0
Müller's Algo.	M_p	2743	3197	3254
	Mc_p	1342	1521	1545
	I_p	1	1	1

Table 6.3: Number of operations in $\mathbb{F}_p$ for square roots in $\mathbb{F}_{q^2}$, $q = p^3$, $p \equiv 3 \pmod 4$

Parameter		$u = -(2^{62} + 2^{55} + 1)$	$p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	$u = 2^{63} + 2^9 + 2^8 + 2^6 + 2^4 + 2^3 + 1$
Bit length of p		254	256	258
s: $p^6 - 1 = 2^s t$, t odd		3	97	4
Algo. 9	M_p	7686	10830	8682
	Mc_p	3693	4921	4091
	I_p	0	0	0
Complex Algo.	M_p	3698	4926	4096
	Mc_p	1229	1589	1345
	I_p	1	1	1
Tonelli-Shanks	M_p	31993	62886	32368
	Mc_p	14469	29715	14650
	I_p	0	0	0
Müller's Algo.	M_p	49299	47032	47033
	Mc_p	19838	20054	20144
	I_p	1	1	1

Table 6.4: Number of operations in $\mathbb{F}_p$ for square roots in $\mathbb{F}_{q^2}$, $q = p^6$, $p \equiv 1 \pmod 4$

Parameter		$p = 2^{224} - 2^{96} + 1$	$u = 2^{62} - 2^{54} + 2^{44}$	$u = 2^{63} - 2^{49}$
Bit length of p		224	254	256
s: $p^{12} - 1 = 2^s t$, t odd		98	47	52
Algo. 10	M_p	28487	23203	24262
	Mc_p	11665	10158	10586
	I_p	1	0	0
Complex Algo.	M_p	34279	20436	23293
	Mc_p	11045	7559	8676
	I_p	7	3	3
Tonelli-Shanks	M_p	265861	213111	222382
	Mc_p	115155	91636	95669
	I_p	0	0	0
Müller's Algo.	M_p	243036	274681	278824
	Mc_p	102438	115789	117569
	I_p	1	1	1

7 Concluding Remarks

7.1 Conclusions

We presented a comprehensive study of the DLP algorithms introduced in 2013 and their different variants by devising a convenient framework and tools for performing concrete analyses. This analysis allowed us to assess the real impact these algorithms have on the security of Type 1 pairing-based cryptography.

We first showed that using the new algorithms on the finite fields $\mathbb{F}_{3^{6\cdot 509}}$, $\mathbb{F}_{2^{12\cdot 367}}$ and $\mathbb{F}_{2^{4\cdot 3041}}$, the supersingular curves with embedding degree 6, 12 and 4 defined, respectively, over $\mathbb{F}_{3^{509}}$, $\mathbb{F}_{2^{367}}$ and $\mathbb{F}_{2^{3041}}$ that had been considered for implementing Type 1 pairing-based cryptosystems at the 128-bit and 192-bit security levels in fact provides only a significantly lower level of security.

We then examined the effectiveness of the new algorithms when incorporating the polynomial representation of Granger and Zumbrägel [64] for computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 1429}}$ and subsequently computed discrete logarithms in the 1303-bit finite field $\mathbb{F}_{3^{6\cdot 137}}$ and the 1551-bit finite field $\mathbb{F}_{3^{6\cdot 163}}$ using a Magma implementation of Joux's algorithm.

Next, we showed that the practical improvements from [60], [61] and [82] yield lower bounds on the estimates for computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 509}}$. The estimates have plummeted to such an extent that we were able to compute discrete logarithms in the order-r subgroup of $\mathbb{F}^*_{3^{6\cdot 509}}$, where $r = (3^{509} - 3^{255} + 1)/7$ is an 804-bit prime, in about 220 CPU years. Moreover, we used techniques from [60] to further weaken the field $\mathbb{F}_{3^{6\cdot 1429}}$ for Type 1 pairing-based cryptography.

The concrete analyses and experimental computations performed in this study allow us to conclude that Type 1 pairings are no longer suitable for pairing-based cryptography, and thus this marks the sad end to constructive uses of these pairings in cryptography.

Finally, we introduced two novel algorithms for computing square roots in even-degree extension fields. Our complexity analysis and experimental comparisons suggest that the new algorithms are good candidates as alternatives to the previous fastest algorithm, namely, Scott's complex method [113]. We also presented a fast procedure for testing quadratic residuosity in extension fields.

7.2 Future work

Solving the DLP in $\mathbb{F}_{2^{4\cdot 1223}}$. After the effective discrete logarithm computation projects for the fields $\mathbb{F}_{3^{6\cdot 137}}$, $\mathbb{F}_{3^{6\cdot 163}}$ and $\mathbb{F}_{3^{6\cdot 509}}$, another challenge can be to compute discrete logarithms in the fields $\mathbb{F}_{2^{4\cdot 1223}}$. In fact, the embedding degree 4 elliptic curve $E_2 : y^2 + y = x^3 + x$ over $\mathbb{F}_{2^{1223}}$ with $|E(\mathbb{F}_{2^{1223}})| = 5r$ where r is a 1221-bit prime had been proposed [9] for pairing-based cryptography under the argument that the finite field $\mathbb{F}_{2^{4\cdot 1223}}$ offers approximately 128 bits of

security against attacks on the DLP by Coppersmith's algorithm. Granger et al. [60] analyzed the security level of the DLP in this field and found that it offers a much lower level of security, namely 2^{59} multiplications modulo a 1221-bit prime. However, the proposed plan of attack is still infeasible in practice because of the high cost of the linear algebra. Exploiting the latest improvements on the new DLP algorithms, in particular partitioning the linear algebra into several smaller subproblems, one can derive estimates allowing a reasonable effort for practical discrete logarithm computations in $\mathbb{F}_{2^{4\cdot 1223}}$.

Improving the Gröbner bases descent. The bilinear systems arising from Gröbner bases descents need to be studied in a more specific way according to their particular shape. By doing this, one can find more adapted variants of the existing Gröbner basis extractor algorithms for the purpose of accelerating the Gröbner bases descent and therefore allowing descents from higher degrees. In this scenario, one expects to automatically have a larger range of practical applicability of the new algorithms. The mentioned study can be done by first trying to efficiently implement Faugere's F4 algorithm in a low-level language, such as C, since only very few researchers have efficient implementations of this algorithm, which they typically want to keep private (this is the reason way we have to always use Magma to perform the Gröbner bases descent).

Computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 709}}$. The integer 709 is the largest prime n smaller than 729 for which there is a supersingular curve E over $\mathbb{F}_{3^n}$ with $|E(\mathbb{F}_{3^n})|$ a prime. For $n = 709$, the size of the curve $E(\mathbb{F}_{3^n})$ is a 1124-bit prime r, and the field $\mathbb{F}_{3^{6\cdot n}}$ is a 6743-bit field. Guillevic [65] recently proposed an alternative to continued-fractions descent. Using this new descent technique, our preliminary estimates show that the discrete logarithms in the order r subgroup of $\mathbb{F}_{3^{6\cdot 709}}$ can be computed in approximately the same time as we expended on our $\mathbb{F}_{3^{6\cdot 509}}$ computation.

How to render $\mathbb{F}_{3^{6\cdot 1429}}$ and $\mathbb{F}_{2^{4\cdot 3041}}$ breakable in practice. An interesting avenue for future research would be to investigate whether the new algorithms and their implementations can be improved to the extent that discrete logarithms in $\mathbb{F}_{3^{6\cdot 1429}}$ and $\mathbb{F}_{2^{4\cdot 3041}}$ can actually be computed in practice. Note that every effective discrete logarithm computation in a finite field of high level of difficulty is very likely to open new doors of interesting improvements. Thus, the objective of trying to solve the DLP in larger and larger finite fields is a good means to achieve better algorithms for this problem.

7.3 List of publications

- G. Adj, A. Menezes, T. Oliveira and F. Rodríguez-Henríquez, "Weakness of $\mathbb{F}_{3^{6\cdot 509}}$ for discrete logarithm cryptography", *Pairing-Based Cryptography – Pairing 2013*, LNCS 8365 (2014), 20–44.

- G. Adj, A. Menezes, T. Oliveira and F. Rodríguez-Henríquez, "Weakness of $\mathbb{F}_{3^{6\cdot 1429}}$ and $\mathbb{F}_{2^{4\cdot 3041}}$ for discrete logarithm cryptography", *Finite Fields and Their Applications*, 32 (2015), 148–170.

- G. Adj, A. Menezes, T. Oliveira and F. Rodríguez-Henríquez, "Computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 137}}$ and $\mathbb{F}_{3^{6\cdot 163}}$ using Magma", *Arithmetic of Finite Fields – WAIFI 2014*, LNCS 9061 (2014), 3–22.

- G. Adj and F. Rodríguez-Henríquez, "Square root computation over even extension fields", *IEEE Transactions on Computers*, 63 (2014), 2829–2841.

Bibliography

[1] Abacus Supercomputer – Cinvestav, `http://www.abacus.cinvestav.mx/`.

[2] J. Adikari, M. Anwar Hasan and C. Negre, "Towards faster and greener cryptoprocessor for eta pairing on supersingular elliptic curve over $\mathbb{F}_{2^{1223}}$", *Selected Areas in Cryptography – SAC 2012*, LNCS 7707 (2013), 166–183.

[3] G. Adj, A. Menezes, T. Oliveira and F. Rodríguez-Henríquez, "Weakness of $\mathbb{F}_{3^{6\cdot 509}}$ for discrete logarithm cryptography", *Pairing-Based Cryptography – Pairing 2013*, LNCS 8365 (2014), 20–44.

[4] G. Adj, A. Menezes, T. Oliveira and F. Rodríguez-Henríquez, "Weakness of $\mathbb{F}_{3^{6\cdot 1429}}$ and $\mathbb{F}_{2^{4\cdot 3041}}$ for discrete logarithm cryptography", *Finite Fields and Their Applications*, 32 (2015), 148–170.

[5] G. Adj, A. Menezes, T. Oliveira and F. Rodríguez-Henríquez, "Computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 137}}$ and $\mathbb{F}_{3^{6\cdot 163}}$ using Magma", *Arithmetic of Finite Fields – WAIFI 2014*, LNCS 9061 (2014), 3–22.

[6] G. Adj and F. Rodríguez-Henríquez, "Square root computation over even extension fields", *IEEE Transactions on Computers*, 63 (2014), 2829–2841.

[7] L. Adleman and M.-D. Huang, "Function field sieve method for discrete logarithms over finite fields", *Information and Computation*, 151 (1999), 5–16.

[8] L. Adleman, "A subexponential algorithm for the discrete logarithm problem with applications to cryptography", *Found. Comp. Sci. Symp. – 20th annual IEEE*, (1979), 55–60.

[9] O. Ahmadi, D. Hankerson and A. Menezes, "Software implementation of arithmetic in $\mathbb{F}_{3^m}$", *International Workshop on Arithmetic of Finite Fields – WAIFI 2007*, LNCS 4547 (2007), 85–102.

[10] D. Aranha, J. Beuchat, J. Detrey and N. Estibals, "Optimal eta pairing on supersingular genus-2 binary hyperelliptic curves", *Topics in Cryptology – CT-RSA 2012*, LNCS 7178 (2012), 98–115.

[11] M. Atiyah and I Macdonald "Introduction to commutative algebra", *Sarat Book House*, (2007).

[12] A. Atkin, "Probabilistic primality testing, summary by F. Morain", *Research Report INRIA*, 1779 (1992), 159–163.

[13] E. Bach and K. Huber, "Note on taking square-roots modulo N", *IEEE Transactions on Information Theory*, 45 (1999), 807–809.

[14] R. Barbulescu, C. Bouvier, J. Detrey, P. Gaudry, H. Jeljeli, E. Thomé, M. Videau and P. Zimmermann, "Discrete logarithm in $GF(2^{809})$ with FFS", *Public Key Cryptography – PKC 2014*, LNCS 8383 (2014), 221–238.

[15] R. Barbulescu and P. Gaudry, personal communication, August 12, 2013.

[16] R. Barbulescu, P. Gaudry, A. Joux and E. Thomé, "A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic: Improvements over FFS in small to medium characteristic", *Advances in Cryptology – EUROCRYPT 2014*, LNCS 8441 (2014), 1–16.

[17] P. Barreto, S. Galbraith, C. Ó hÉigeartaigh and M. Scott, "Efficient pairing computation on supersingular abelian varieties", *Designs, Codes and Cryptography*, 42 (2007), 239–271.

[18] P. Barreto, H. Kim, B. Lynn and M. Scott, "Efficient algorithms for pairing-based cryptosystems", *Advances in Cryptology – CRYPTO 2002*, LNCS 2442 (2002), 354–368.

[19] P. Barreto, B. Lynn and Michael Scott, "Constructing elliptic curves with prescribed embedding degrees", *Security in Communication Networks – SCN 2002*, LNCS 2576 (2003), 257–267.

[20] P. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order, *Selected Areas in Cryptography – SAC 2005*, LNCS 3897 (2005), 319–331.

[21] P. Barreto and J. Voloch, "Efficient computation of roots in finite fields", *Des. Codes Cryptography*, 39 (2006), 275–280.

[22] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor", *Advances in Cryptology – CRYPTO '86*, LNCS 263 (1987), 311–323.

[23] N. Benger and M. Scott, "Constructing tower extensions of finite fields for implementation of pairing-based cryptography", *Arithmetic of Finite Fields – WAIFI 2010*, LNCS 6087 (2010), 180–195.

[24] D. Bernstein, "How to find small factors of integers", manuscript, 2002; available at `http://cr.yp.to/papers/sf.pdf`.

[25] J. Beuchat, J. Detrey, N. Estibals, E. Okamoto and F. Rodríguez-Henríquez, "Fast architectures for the η_T pairing over small-characteristic supersingular elliptic curves", *IEEE Transactions on Computers*, 60 (2011), 266–281.

[26] J. Beuchat, J. González-Díaz S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez and Tadanori Teruya, "High-speed software implementation of the optimal Ate pairing over Barreto-Naehrig curves", *Pairing-Based Cryptography – Pairing 2010*, LNCS 6487 (2010), 21–39.

[27] J. Beuchat, E. López-Trejo, L. Martínez-Ramos, S. Mitsunari and F. Rodríguez-Henríquez, "Multi-core implementation of the Tate pairing over supersingular elliptic curves", *Cryptology and Network Security – CANS 2009*, LNCS 5888 (2009), 413–432.

[28] I. Blake, R. Fuji-Hara, R. Mullin and S. Vanstone, "Computing logarithms in finite fields of characteristic two", *SIAM Journal on Algebraic and Discrete Methods*, 5 (1984), 276–285.

[29] A. Bluher, "On $x^{q+1}+ax+b$", *Finite Fields and Their Applications*, 10 (2004), 285–305.

[30] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing", *Advances in Cryptology – CRYPTO 2001*, LNCS 2139 (2001), 213–229.

[31] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing", *Journal of Cryptology*, 17 (2004), 297–319.

[32] D. Boneh and H. Shacham, "Group signatures with verifier-local revocation" *ACM Conference on Computer and Communications Security – CCS 2004*, (2004), 168 – 177.

[33] I. Canales Martinez, "Implementación eficiente de prueba de suavidad para polinomios", *Tesis de Maestría – CINVESTAV-IPN 2015*, available at `http://delta.cs.cinvestav.mx/~francisco/Thesis_IAC.pdf`.

[34] S. Chatterjee, D. Hankerson, E. Knapp and A. Menezes, "Comparing two pairing-based aggregate signature schemes", *Des. Codes Cryptography*, 55 (2010), 141–167.

[35] S. Chatterjee, D. Hankerson and A. Menezes, "On the efficiency and security of pairing-based protocols in the Type 1 and Type 4 settings", *International Workshop on Arithmetic of Finite Fields – WAIFI 2010*, LNCS 6087 (2010), 114–134.

[36] Q. Cheng, D. Wan and J. Zhuang, "Traps to the BGJT-algorithm for discrete logarithms", *LMS Journal of Computation and Mathematics* 17 (2014), 218–229.

[37] C. Chuengsatiansup, M. Naehrig, P. Ribarski, P. Schwabe, "PandA: pairings and arithmetic", *Pairing-Based Cryptography – Pairing 2013*, LNCS 8365 (2014), 229–250.

[38] M. Cipolla, "Un metodo per la risoluzione della congruenza di secondo grado", *Rend. Accad. Sci. Fis. Mat. Napoli*, 9 (1903), 154–163.

[39] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen and F. Vercauteren, "Handbook of elliptic and hyperelliptic curve cryptography", *Discrete Mathematics and Its Applications – CRC Press*, (2005).

[40] D. Coppersmith, "Fast evaluation of logarithms in fields of characteristic two", *IEEE Transactions on Information Theory*, 30 (1984), 587–594.

[41] D. Coppersmith, "Solving homogeneous linear equations over $GF(2)$ via block Wiedemann algorithm", *Mathematics of Computation*, 62 (1994), 333–350.

[42] The Cunningham Project, `http://homes.cerias.purdue.edu/~ssw/cun/`.

[43] W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory*, 22 (1976), 644–654.

[44] J. Doliskani and E. Schost, "Taking roots over high extensions of finite fields", *CoRR*, 1110 (2011), 4350.

[45] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Transactions on Information Theory*, 31 (1985), 469–472.

[46] N. Estibals, "Compact hardware for computing the Tate pairing over 128-bit-security supersingular curves", *Pairing-Based Cryptography – Pairing 2010*, LNCS 6487 (2010), 397–416.

[47] J. Faugère, "A new efficient algorithm for computing Gröbner bases (F_4)", *Journal of Pure and Applied Algebra*, 139 (1999), 61–88.

[48] J. Faugère, "A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5)", *International Symposium on Symbolic and Algebraic Computation – ISSAC 2002* (2002), 75–83.

[49] G. Frey and H. Rück, "A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves", *Mathematics of Computation*, 62 (1994), 865–874.

[50] P. Friedland, "Algorithm 312: Absolute value and square root of a complex number", *Commun. ACM*, 10 (1967), 665–.

[51] S. Galbraith, "Supersingular curves in cryptography", *Advances in Cryptology – ASIACRYPT 2001*, LNCS 2248 (2001), 495–513.

[52] S. Galbraith, K. Harrison and D. Soldera, "Implementing the Tate pairing", *Algorithmic Number Theory – ANTS 2002*, LNCS 2369 (2002), 324–337.

[53] S. Galbraith, K. Paterson and N. Smart, "Pairings for cryptographers", *Discrete Applied Mathematics*, 156 (2008), 3113–3121.

[54] C. F. Gauss, "Untersuchungen über höhere Arithmetik", *Chelsea publishing company*, second edition, reprinted, New York (1981).

[55] W. Geiselmann, A. Shamir, R. Steinwandt and E. Tromer, "Scalable hardware for sparse systems of linear equations, with applications to integer factorization", *Cryptographic Hardware and Embedded Systems – CHES 2005*, LNCS 3659 (2005), 131–146.

[56] C. Pereira Geovandro, M. Simplício Jr., M. Naehrig and P. Barreto, "A family of implementation-friendly BN elliptic curves", *Journal of Systems and Software*, 84 (2011), 1319–1326.

[57] F. Göloğlu, R. Granger, G. McGuire and J. Zumbrägel, "On the function field sieve and the impact of higher splitting probabilities: Application to discrete logarithms in $\mathbb{F}_{2^{1971}}$", *Advances in Cryptology – CRYPTO 2013*, LNCS 8043 (2013), 109–128.

[58] F. Göloğlu, R. Granger, G. McGuire and J. Zumbrägel, "Solving a 6120-bit DLP on a desktop computer", *Selected Areas in Cryptography – SAC 2013*, LNCS 8282 (2014), 136–152.

[59] R. Granger, T. Kleinjung and J. Zumbrägel, "Breaking '128-bit secure' supersingular binary curves (or how to solve discrete logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$ and $\mathbb{F}_{2^{12\cdot 367}}$)", available at `http://eprint.iacr.org/2014/119`.

[60] R. Granger, T. Kleinjung and J. Zumbrägel, "Breaking '128-bit secure' supersingular binary curves (or how to solve discrete logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$ and $\mathbb{F}_{2^{12\cdot 367}}$)", *Advances in Cryptology – CRYPTO 2014*, Part II, LNCS 8617 (2014), 126–145.

[61] R. Granger, T. Kleinjung and J. Zumbrägel, "On the powers of 2", available at `http://eprint.iacr.org/2014/300`.

[62] 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 (2005), 852–860.

[63] R. Granger, D. Page and M. Stam, "On small characteristic algebraic tori in pairing-based cryptography", *LMS Journal of Computation and Mathematics*, 9 (2006), 64–85.

[64] R. Granger and J. Zumbrägel, "On the security of supersingular binary curves", presentation at ECC 2013, September 16 2013.

[65] A. Guillevic, "Faster individual discrete logarithms in non-prime finite fields with the NFS and FFS algorithms", available at `http://eprint.iacr.org/2016/684`.

[66] D. Han, D. Choi and H. Kim, "Improved computation of square roots in specific finite fields", *IEEE Transaction on Computers*, 58 (2009), 188–196.

[67] D. Hankerson, A. Menezes and M. Scott, "Software implementation of pairings", *Identity-Based Cryptography – Cryptology and Information Security Series*, 12 (2009), 188–206.

[68] R. Hartshorne, "Algebraic geometry", *Graduate Texts in Mathematics – Springer New York*, (2013).

[69] T. Hayashi, T. Shimoyama, N. Shinohara and T. Takagi, "Breaking pairing-based cryptosystems using η_T pairing over $GF(3^{97})$", *Advances in Cryptology – ASIACRYPT 2012*, LNCS 7658 (2012), 43–60.

[70] T. Helleseth and A. Kholosha, "$x^{2^l+1}+x+a$ and related affine polynomials over GF(2^k)", *Cryptogr. Commun.*, 2 (2010), 85–109.

[71] M. Hellman and J. Reyneri, "Fast computation of discrete logarithms in GF(q)", *Advances in Cryptography – CRYPTO 82*, (1983), 3–13.

[72] IEEE, "IEEE P1363-2000 Draft Standard for Traditional Public-Key Cryptography", 1363 (2006).

[73] T. Itoh, O. Teechai and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases", *Information and Computation*, 78 (1988), 171-177.

[74] H. Jeljeli, "Accelerating iterative SpMV for discrete logarithm problem using GPUs", *Arithmetic of Finite Fields – WAIFI 2014*, LNCS 9061 (2014), 25–44.

[75] A. Joux, "A one round protocol for tripartite Diffie-Hellman", *Journal of Cryptology*, 17 (2004), 263–276.

[76] A. Joux, "Faster index calculus for the medium prime case: Application to 1175-bit and 1425-bit finite fields", *Advances in Cryptology – EUROCRYPT 2013*, LNCS 7881 (2013), 177–193.

[77] A. Joux, "A new index calculus algorithm with complexity $L(1/4 + o(1))$ in very small characteristic", *Selected Areas in Cryptography – SAC 2013*, LNCS 8282 (2014), 355–379.

[78] A. Joux, "Discrete logarithm in $GF(2^{6128})$", Number Theory List, May 21 2013.

[79] A. Joux and R. Lercier, "The function field sieve is quite special", *Algorithmic Number Theory – ANTS 2002*, LNCS 2369 (2002), 431–445.

[80] A. Joux and R. Lercier, "Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the Gaussian integer method", *Mathematics of Computation*, 72 (2003), 953–967.

[81] A. Joux and R. Lercier, "The function field sieve in the medium prime case" *Advances in Cryptology – EUROCRYPT 2006*, LNCS 4004 (2006), 254–270.

[82] A. Joux and C. Pierrot, "Improving the polynomial time precomputation of Frobenius representation discrete logarithm algorithms", *Advances in Cryptology – ASIACRYPT 2014*, LNCS 8873 (2014), 378–397.

[83] E. Kachisa, E. Schaefer and Michael Scott, "Constructing Brezing-Weng pairing-friendly elliptic curves using elements in the cyclotomic field", *Pairing-Based Cryptography – Pairing 2008*, LNCS 5209 (2008), 126–135.

[84] H. Kato, Y. Nogami and Y. Morikawa, "A high-speed square root algorithm for extension fields", *Memoirs of the Faculty of Engineering, Okayama University*, 43 (2009), 99–107.

[85] H. Katou, F. Wang, Y. Nogami and Y. Morikawa, "A high-speed square root algorithm in extension fields", *Information Security and Cryptology - ICISC 2006*, LNCS 4296 (2006), 94–106.

[86] N. Koblitz, "A course in number theory and cryptography", *Graduate Texts in Mathematics – Springer New York*, 2012.

[87] N. Koblitz and A. Menezes, "Pairing-based cryptography at high security levels", *Cryptography and Coding – IMA 2005*, LNCS 3796 (2005), 13–36.

[88] N. Koblitz and A. Menezes, Y. Wu and R. Zuccherato, "Algebraic aspects of cryptography", *Algorithms and Computation in Mathematics – Springer Berlin Heidelberg*, (2012).

[89] F. Kong, Z. Cai, J. Yu and D. Li, "Improved generalized Atkin algorithm for computing square roots in finite fields", *Information Processing Letters*, 98 (2006), 1–5.

[90] T. Kleinjung "Discrete Logarithms in $GF(2^{1279})$", Number Theory List, October 17 2014.

[91] A. Knopfmacher and J. Knopfmacher, "Counting irreducible factors of polynomials over a finite field", *Discrete Mathematics*, 112 (1993), 103–118.

[92] M. Kraitchik, "Théorie des nombres", *Gauthier-Villars – Paris*, (1922).

[93] B. LaMacchia and A. Odlyzko, "Solving large sparse linear systems over finite fields", *Advances in Cryptology – CRYPTO '90*, LNCS 537 (1991), 109–133.

[94] A. Lenstra, "Unbelievable security: Matching AES security using public key systems", *Advances in Cryptology – ASIACRYPT 2001*, LNCS 2248 (2001), 67–86.

[95] A. Lenstra, A. Shamir, J. Tomlinson and E. Tromer, "Analysis of Bernstein's factorization circuit", *Advances in Cryptology – ASIACRYPT 2002*, LNCS 2501 (2002), 1–26.

[96] R. Lidl and H. Niederreiter "Finite Fields", *Encyclopedia of Mathematics and its Applications – Cambridge University Press*, 20 (1997).

[97] S. Lindhurst, "An analysis of Shanks's algorithm for computing square roots in finite fields", *CRM Proc. and Lecture Notes*, 19 (1999), 231–242.

[98] Magma Computational Algebra System, `http://magma.maths.usyd.edu.au/magma/`.

[99] Maple 17, `http://www.maplesoft.com/products/maple/`.

[100] A. Menezes, T. Okamoto and S. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field", *IEEE Transactions on Information Theory*, 39 (1993), 1639–1646.

[101] R. Merkle, "Secrecy, authentication, and public key systems", *Ph.D. dissertation, Dept. of Electrical Engineering – Stanford Univ.* (1979).

[102] J. Miller, "On factorization, with a suggested new approach", *Math. Comp.* 29 (1975), 155–172.

[103] V. Miller, "Use of elliptic curves in cryptography", *Advances in Cryptology – CRYPTO '85*, LNCS 218 (1985), 417–426.

[104] M. Morrison and J. Brillhart, "A method of factoring and the factorization of F_7", *Math. Comp.* 29 (1975), 183–205.

[105] G. Mullen and D. Panario, "Handbook of finite fields", *Discrete Mathematics and Its Applications – CRC Press*, (2013).

[106] S. Müller, "On the computation of square roots in finite fields", *Des. Codes Cryptography*, 31 (2004), 301–312.

[107] NIST, "Digital Signature Standard", *FIPS Publication*, 186-2 (2000).

[108] D. Page, N. Smart and F. Vercauteren, "A comparison of MNT curves and supersingular curves", *Applicable Algebra in Engineering, Communication and Computing*, 17 (2006), 379–392.

[109] J. Pollard, "Monte Carlo methods for index computation mod p", *Mathematics of Computation*, 32 (1978), 918–924.

[110] A. Sahai and B. Waters, "Fuzzy identity-based encryption", *Advances in Cryptology – EUROCRYPT 2005*, LNCS 3494 (2005), 457–473.

[111] R. Sakai, K. Ohgishi and M. Kasahara, "Cryptosystems based on pairing", *The 2000 Symposium on Cryptography and Information Security – Japan*, 28 (2000), 45–26.

[112] R. Schoof, "Elliptic curves over finite fields and the computation of square roots mod p", *Mathematics of Computation*, 44 (1985), 483–494.

[113] M. Scott, "Implementing cryptographic pairings over Barreto-Naehrig curves", *Pairing-Based Cryptography – Pairing 2007*, LNCS 4575 (2007), 177–196.

[114] D. Shanks, "Five number-theoretic algorithms", *Proceedings of the second Manitoba conference on numerical mathematics*, (1972), 51–70.

[115] N. Shinohara, T. Shimoyama, T. Hayashi and T. Takagi, "Key length estimation of pairing-based cryptosystems using η_T pairing", *Information Security Practice and Experience – ISPEC 2012*, LNCS 7232 (2012), 228–244.

[116] V. Shoup, "Lower bounds for discrete logarithms and related problems", *Advances in Cryptology – EUROCRYPT 1997*, (1997), 256–266.

[117] J. Silverman, "The Arithmetic of elliptic curves", *Graduate Texts in Mathematics – Springer New York*, (2009).

[118] R. Swan, "Factorization of polynomials over finite fields", *Pacific J. of Math.* 12 (1962), 1099–1106.

[119] G. Tornaría, "Square roots modulo p", *Theoretical Informatics – LATIN 2002*, LNCS 2286 (2002), 430–434.

[120] A. Tonelli, "Bemerkung uber die Auflosung quadratischer Congruenzen", *Götinger Nachrichten*, (1891), 344–346.

[121] F. Wang, Y. Nogami and Y. Morikawa, "An efficient square root computation in finite fields GF$\left(p^{2^d}\right)$", *IEICE Transactions*, 88 (2005), 2792–2799.

[122] L. Washington, "Elliptic curves: number theory and cryptography", *Discrete Mathematics and Its Applications – CRC Press*, (2003).

[123] A. Western and J. Miller, "Tables of indices and primitive Roots", *Royal Society Mathematical – Cambridge Univ. Press* (1968).

[124] D. Wiedemann, "Solving sparse linear equations over finite fields", *IEEE Transactions on Information Theory*, 32 (1986), 54–62.