Compressing Elements in Discrete Logarithm Cryptography

Philip Nicholas James Eagle, Esq.

Submitted in total fulfilment of the requirements
of the degree of Philosophiæ Doctor

June 2008

Information Security Group
Royal Holloway College, University of London

Abstract

In the modern world, the ubiquity of digital communication is driven by the constantly evolving world of cryptography. Consequently one must efficiently implement asymmetric cryptography in environments which have limited resources at their disposal, such as smart–cards, ID cards, vehicular microchips and many more. It is the primary purpose of this thesis to investigate methods for reducing the bandwidth required by these devices.

Part I of this thesis considers compression techniques for elliptic curve cryptography (ECC). We begin this by analysing how much data is actually required to establish domain parameters for ECC. Following the widely used cryptographic standards (for example: SEC 1), we show that naïvely implemented systems use extensively more data than is actually required and suggest a flexible and compact way to better implement these. This is especially of use in a multi–curve environment. We then investigate methods for reducing the inherent redundancy in the point representation of Koblitz systems; a by–product of the best known Pollard–ρ based attacks by Wiener & Zuccherato and Gallant, Lambert & Vanstone. We present methods which allow such systems to operate (with a high confidence) as efficiently as generic ones whilst maintaining all of their computational advantages.

Figure 1: Royal Holloway

In Part II we investigate using disguised algebraic tori, LUC and XTR as candidates for black–box groups. It is well known that the specific representation of groups affect their suitability to be used for cryptography. Black–box group cryptography, where little or nothing is known about the underlying group structure exploits this idea. Such group representations could potentially lead to new applications. This work is motivated by the trapdoor DDH groups of Dent & Galbraith based on earlier work by Frey. We detail specific attacks to undisguise these groups and make comments on future directions in this area.

Acknowledgements

A great deal of heartfelt thanks goes to my supervisor Dr. S. D. Galbraith. It is impossible to articulate the depth of my debt to Steven, who has always been sagacious in his guidance and tireless in his support. Thank you.

To my parents

Contents

Abstract i

Acknowledgements ii

1 Motivation 1

2 Cryptographic Background 4
2.1 Complexity Theory . . . 4
2.1.1 Complexity Notation . . . 4
2.1.2 Complexity Classes . . . 5
2.2 Asymmetric Cryptography . . . 5
2.2.1 Computational Security and Hard Problems . . . 6
2.3 The Discrete Logarithm Problem . . . 6
2.4 Attacks on the DLP: Pohlig–Hellman . . . 7
2.5 Cryptographic Schemes Based on the DLP . . . 7
2.5.1 DHKA–Protocol . . . 7
2.5.2 ElGamal–Protocol . . . 8

3 Mathematical Background: Finite Fields 9
3.1 The Structure of Finite Fields . . . 9
3.2 Representing Finite Fields . . . 11
3.2.1 Polynomial Representation . . . 11
3.2.2 Normal Representation . . . 12
3.3 Computing Isomorphisms Between Finite Fields . . . 12

I Compression 14

4 Background 15
4.1 Elliptic Curves . . . 15
4.1.1 Addition Law for Curve Points . . . 17
4.1.2 Point Compression . . . 18
4.1.3 Seroussi's Point Compression for Curves $E/\mathbb{F}_{2^n}$ . . . 19
4.2 Koblitz Curves . . . 20
4.3 Edwards Curves . . . 21
4.4 Security Parameters . . . 22

5 Compression of Elliptic Curve Domain Parameters 23
5.1 Motivation . . . 23
5.2 Domain Parameters . . . 24
5.2.1 SEC 1: Elliptic Curve Domain Parameters . . . 24
5.2.2 Naïve Representational Bit–Size for $\mathcal{V}$ in SEC 1 . . . 25
5.3 Previous Research: Smart . . . 28
5.4 Reducing Redundancy in the Definition of $\mathcal{V}$ . . . 29
5.4.1 Establishing the Order ℓ for General Finite Fields . . . 29
5.4.2 The Modified Cofactor . . . 30
5.5 Excluding Redundancy from $\mathcal{V}$ for SEC 1 . . . 31
5.6 Subfield Curves under SEC 1 . . . 32

5.6.1 Establishing the Order ℓ when using Subfield Curves . . . 32
5.6.2 Performance Considerations . . . 34
5.6.3 Modified Trace Unsigning when Using Verification . . . 35
5.6.4 The Modified Cofactor for Subfield Curves . . . 37
5.7 Discussions & Conclusions . . . 37

6 Compact Domain Parameters 39
6.1 Motivation . . . 39
6.2 Previous Research: Smart . . . 40
6.3 Previous Research: Brown, Myers & Solinas . . . 41
6.4 Compact Domain Parameters $\mathcal{V}$ . . . 43
6.5 Defining the Field, $\mathcal{F} \in \mathcal{V}$ . . . 44
6.5.1 Compactly Representing Special Primes; Hence $\mathbb{F}_p$ . . . 44
6.5.2 Representing Extension Fields . . . 50
6.5.3 Representing Extension Fields: Characteristic Two . . . 50
6.5.4 Representing Extension Fields; for Koblitz Curves . . . 52
6.5.5 Representing Extension Fields: Optimal Extension Fields 53
6.6 Specifying the Curve . . . 55
6.6.1 The Orders of the Curve: $\#E$ and ℓ . . . 56
6.6.2 Choosing the Coefficient b for Weierstraß Forms . . . 57
6.6.3 Probability of Occurring Orders of $\#E$ . . . 57
6.6.4 Choosing the Coefficient d for Edwards Forms . . . 61
6.6.5 Summary . . . 62
6.7 Specifying Base Points . . . 63
6.8 Definition and Generation of Multi–Curve CDPs . . . 64
6.8.1 For Prime Fields $\mathbb{F}_p$ . . . 64
6.8.2 For Binary Fields $\mathbb{F}_{2^n}$ . . . 66
6.8.3 For Koblitz Curves $E/\mathbb{F}_{2^n}$ . . . 66
6.9 Discussions & Conclusions . . . 67

7 Point Compression for Koblitz Curves 69
7.1 Motivation . . . 69
7.1.1 Overhead in Koblitz Systems . . . 70
7.2 Reducing Bandwidth: Point Compression . . . 71
7.3 Compression & Decompression Algorithms . . . 71
7.4 Compressing Koblitz Abscissæ: Theory . . . 73
7.4.1 Equivalence Classes . . . 73
7.4.2 Compressing Koblitz Abscissæ up to Rotation . . . 74
7.5 DHKA using Compressed Points . . . 76
7.6 Bandwidth Reduction: Theoretical Expectations . . . 77
7.7 Bandwidth Reduction: Practical Results . . . 78
7.8 Using a Variable Length Communication Model . . . 79
7.9 Discussion of Practical Results & Conclusions . . . 79

II Disguising 81

8 Background 82
8.1 Subgroups and Algebraic Tori . . . 82
8.2 Rubin & Silverberg's Parameterisation of T_2 . . . 83

8.3 Trace Based Cryptosystems: LUC . . . 85
8.4 Trace Based Cryptosystems: XTR . . . 87

9 Disguising Objects & Black–Boxes 89
9.1 Introduction to Black–Box Groups . . . 89
9.1.1 Generic Algorithms . . . 89
9.1.2 Black–Box Groups . . . 90
9.1.3 Security . . . 92
9.1.4 Computing Maps Between Representations . . . 93
9.1.5 Previous Research . . . 93
9.1.6 Our Contribution . . . 94
9.2 Disguising Finite Fields . . . 94
9.2.1 Construction . . . 94
9.2.2 Cryptanalysis of Disguised Finite Fields . . . 95
9.2.3 Attack Algorithm . . . 97
9.3 Disguising Using Affine Transformations U . . . 98
9.4 Disguising Tori . . . 99
9.4.1 Previous Research: Galbraith's Disguise of T_2 . . . 99
9.4.2 Construction . . . 100
9.4.3 DHKA using Galbraith's Disguised T_2 . . . 101
9.4.4 Cryptanalysis of Galbraith's Construction . . . 101
9.4.5 Disguising T_2 — A New Approach . . . 103
9.4.6 Construction . . . 103
9.4.7 Examples . . . 105
9.4.8 Cryptanalysis of Our Disguised T_2 . . . 106
9.4.9 Disguising Higher–Degree Tori . . . 107
9.5 Disguising Trace–Based Methods: LUC . . . 107
9.5.1 Construction . . . 108
9.5.2 Cryptanalysis of Disguised LUC . . . 109
9.6 Disguising Trace–Based Methods: XTR . . . 110
9.6.1 Construction . . . 110
9.6.2 Cryptanalysis of Disguised XTR . . . 111
9.7 Discussions & Remarks . . . 112

III Appendices 114

A Detailed Methods 115
A.1 SEC 1 Parameter Representation . . . 115
A.1.1 SEC 1: Bit–String–to–Octet–String conversion . . . 115
A.1.2 SEC 1: Representing Integers . . . 116
A.1.3 SEC 1: Representing Finite Field elements . . . 116
A.1.4 SEC 1: Representing Elliptic Curve Points . . . 117

B Tables 119
B.1 Prime Tables . . . 119
B.2 Miscellaneous Tables . . . 131

List of Tables

5.1 SEC 1: Example Bit Sizes for $\mathcal{V}$. 27

6.1 Special Prime Families. 44
6.2 NIST Key–Sizes. 44
6.3 Prime Family Densities. 45
6.4 Computed Type–1, 3 & 4 Maximal Encoded Element Bit–Sizes. . 46
6.5 Computed Type–6 & 8 Maximal Encoded Element Bit–Sizes. . . 47
6.6 Computed IR δ to $4d.p.$ 48
6.7 Values (n, d) and (n, d_2, d_1, d_0) for Low–Weight Prime Order Polynomials of Weight 3 and 5 respectively. 51
6.8 Relevant Degrees of n for Koblitz Curves. 52
6.9 Example Maximal Order Curves $E/\mathbb{F}_{2^{281}}$ for $a = 0$. 61
6.10 Example Maximal Order Curves $E/\mathbb{F}_{2^{281}}$ for $a = 1$. 61

7.1 Additional Bandwidth required for Koblitz Systems 71
7.3 Observed Probability for Koblitz Point Compression 80

B.1 Type–1 Primes p with $4 \leq \lceil \lg p \rceil \leq 64$. 119
B.2 Type–1 Primes p with $160 \leq \lceil \lg p \rceil \leq 224$. 120
B.3 Type–1 Primes p with $224 \leq \lceil \lg p \rceil \leq 256$. 120
B.4 Type–1 Primes p with $256 \leq \lceil \lg p \rceil \leq 572$. 122
B.5 Type–3 Primes p with $4 \leq \lceil \lg p \rceil \leq 64$. 122
B.6 Type–3 Primes p with $160 \leq \lceil \lg p \rceil \leq 224$. 123
B.7 Type–3 Primes p with $224 \leq \lceil \lg p \rceil \leq 256$. 123
B.8 Type–3 Primes p with $256 \leq \lceil \lg p \rceil \leq 572$. 124
B.9 Type–4 Primes p with $4 \leq \lceil \lg p \rceil \leq 64$. 124
B.10 Type–4 Primes p with $160 \leq \lceil \lg p \rceil \leq 224$. 124
B.11 Type–4 Primes p with $224 \leq \lceil \lg p \rceil \leq 256$. 125
B.12 Type–4 Primes p with $256 \leq \lceil \lg p \rceil \leq 572$. 127
B.13 Type–6 Primes p with $4 \leq \lceil \lg p \rceil \leq 64$. 127
B.14 Type–6 Primes p with $160 \leq \lceil \lg p \rceil \leq 224$. 127
B.15 Type–6 Primes p with $224 \leq \lceil \lg p \rceil \leq 256$. 128
B.16 Type–6 Primes p with $256 \leq \lceil \lg p \rceil \leq 572$. 129
B.17 Type–8 Primes p with $4 \leq \lceil \lg p \rceil \leq 64$. 129
B.18 Type–8 Primes p with $160 \leq \lceil \lg p \rceil \leq 224$. 129
B.19 Type–8 Primes p with $224 \leq \lceil \lg p \rceil \leq 256$. 130
B.20 Type–8 Primes p with $256 \leq \lceil \lg p \rceil \leq 572$. 131
B.21 Values of c_q for $q = 2^n$, n Prime. 131

Ya no la quiero, es cierto, pero tal vez la quiero.
Es tan corto el amor, y es tan largo el olvido.

Pablo Neruda (1904–1973)

1 Motivation

Elliptic Curve Cryptography: Kerckhoffs' Law from 1883 states;

"The strength of a cipher should only depend on its key size", [38].

NIST in [6] gives examples of key sizes for equivalent strengths between different asymmetric cryptosystems and groups. These are provided for RSA, finite fields $\mathbb{F}_q$, and elliptic curves over finite fields; $E/\mathbb{F}_q$. Let L be the bit size of the finite field $\mathbb{F}_q$ or the RSA modulus. Then;

	Bit length L		
Bits of Security	RSA	$\mathbb{F}_q$	$E/\mathbb{F}_q$
80	1024	1024	$160 \leq L < 224$
112	2048	2048	$224 \leq L < 256$
128	3072	3078	$256 \leq L < 384$
192	7680	7680	$384 \leq L < 512$
256	15360	15360	$L \geq 512$

Thus if a system requires 80–bits of security, one would need to generate, transmit and store elements of size 1024–bits for the RSA and finite field cases in comparison to approximately 160–bits for the elliptic curve case. This profound difference between equivalent security key–sizes only grows as the security requirements of a system increase. A posteriori, RSA and finite field cryptosystems are unhelpful from a bandwidth point–of–view.

Both bandwidth and the local storage associated with asymmetric cryptography are important: Restricted devices such as smart–cards often have restricted communication speed and internal memory. At the other extreme, servers working in multi–user environments may need to store millions of public–keys.

Cryptography, especially on constrained devices, needs efficient implementation which minimises computational time, memory requirements and communication bandwidth. This is the principal motivation for the deployment of elliptic curve cryptography (ECC) which this thesis considers.

Compressing Elliptic Curve Elements: Part I of this thesis analyses the bandwidth required for ECC. In the initialisation of elliptic curve based cryptosystems one needs to generate, transmit and store system *domain parameters*. These domain parameters, when implemented naïvely as given in standards such as SEC [15], require approximately $5\frac{1}{2}L$–bits to represent.

In Chapter 5 we will explicitly detail the bandwidth required to represent each domain parameter element under the SEC 1 standard. We then suggest

methods to compress these, giving the best case for the communication bandwidth of domain parameters without restricting what the model can represent.

The methods in Chapter 5 apply to general curves and finite fields. However, in practice it is usually recommended to use special field representations and curve equations to give improved performance. In Chapter 6 we propose methods to minimise the bandwidth for domain parameters when using special fields and curve equations. This compact form is particularly bandwidth efficient and thus suitable for multi–curve environments. These results can be applied to many different applications, for example when using the Diffie–Hellman Key–Agreement (DHKA), [20]

In Chapter 7 we consider the message bandwidth of Koblitz systems: Koblitz curves are elliptic curves over $\mathbb{F}_2$ and one uses the group $E(\mathbb{F}_{2^n})$ for ECC. It is well known that Koblitz curves offer various implementational advantages. Both Wiener & Zuccherato [87] and Gallant, Lambert & Vanstone [32] showed that one can accelerate Pollard–ρ type attacks on Koblitz curves using group automorphisms. This implies that when using Koblitz systems, one has a lower security per bit than when using general elliptic curves defined over the same field. Hence for a fixed k–bit security level, Koblitz systems require an increased bandwidth. We present a method to reduce this bandwidth overhead. Subsequently we will show with a low probability of failure one can compress this bandwidth to the expected number of bytes for a given security parameter k when using an analogue of the DHKA.

Disguising Objects & Black–Boxes: The difficulty of a computational problem depends on the specific group representation used. For example; the discrete logarithm problem (DLP) generally requires exponential–time with ECC and polynomial–time for the additive group of integers modulo a prime.

The notion of a *black–box group* (BBG) is designed to model a situation where the representation of group elements and the internal structure of the group operation, do not seem to help solve computational problems in the group. The formal definition of a BBG has group elements represented by random binary strings and allows only oracle access to the group operations. In other words, given two binary strings (corresponding to some group elements g_1 and g_2) the oracle outputs the binary string corresponding to $g_1 \oplus g_2$. Shoup [73] has proven that in a BBG certain computational problems (for example, the DLP and the Diffie–Hellman problem) have exponential complexity. A natural problem is to construct groups which resemble black–box groups, in the sense that the representation of group elements and the formulæ for the group law do not seem to have any structure which can be exploited by algorithms. Such groups could have interesting cryptographic applications.

There has been limited research into finding group representations which behave like BBGs, [19, 26, 28, 51]. One possible approach is to *disguise* a group representation such that it is hard to recover a natural mathematical representation of the original group. Dent & Galbraith in [19] (building on the work of Frey [26]) proposed disguising elliptic curves to get black–box DDH groups with extra functionalities. It is natural to question the security of these systems. As a first step, Galbraith in [28] considered disguised finite fields and tori. In Part II of this thesis, we give cryptanalysis of disguised tori including those with an alternative and novel disguise. We then continue to give a cryptanalysis of

other candidate BBGs created from the trace–based methods of LUC and XTR.

We conclude that disguised tori, LUC and XTR are not secure and that disguising things is general is not a sensible approach to strengthening security.

Original Contribution: This thesis presents backgound material and work of other researchers intermingled with new results. To clarify the novel aspects of this project we now list the main original contributions of the author.

In Chapter 5 we present modified traces and modified cofactors. These enable us to define a curves trace and cofactor in fewer bits than previously published. For general curves over finite fields, these are asymptotically equal to a result by Smart. For subfield curves, these are better. When using point–verification, we give an algorithm that computes the original trace of the full curve using the absolute value of the subfield trace. This lowers the bandwidth by an additional bit. Finally we give a thorough analysis of the exact worst–case domain parameter sizes of SEC 1, which is used as motivation for defining a new model of domain parameter representation.

In Chapter 6 we suggest a more efficient way of specifying domain parameters. We note that in a multi–curve environment, the bandwidth of domain parameters is important and should be minimal. We propose that such a system should be optimised for how it will be used: The range of security parameters over which it will be deployed and how many systems it is required to specify. We analyse special forms of fields which are often used in ECC, and their associated information rate. We use these to present a flexible and compact description of cryptographically useful fields. We present and justify what one requires from an elliptic curve in terms of its order, cofactor and field of definition for efficiency in terms of a security parameter k. We use these results to define a method of generating cryptographically secure ordinary elliptic curves. We show that one can then define a plentiful and suitable amount of such non–special curves in a fraction of the bandwidth used by current propositions. Our model defines domain parameters for prime fields in less than a fifth of the bandwidth normally expected for a given security parameter k. When using binary fields, one asymptotically expects a tenth of the bandwidth required by SEC 1. These results are better than any we know of in the literature. We comment that our generation method is particularly suitable for Edwards curves which have been recently (re)discovered. This leads to parameters which are bandwidth efficient and the best known for curve performance.

In Chapter 7 we present a method to reduce the additional bandwidth required by Koblitz systems for a fixed security parameter k over general ones. Subsequently we show with a low probability of failure one can compress this bandwidth to the expected number of bytes of a general system over the same field. We propose an analogue of the DHKA directly using these compressed points. We provide efficient algorithms for the compression and decompression of such systems. These results are not known to have been published before.

In Chapter 9 we propose disguising certain algebraic objects to form candidate BBGs. We present a novel method of specifying a disguised group law on T_2 and give methods for LUC and XTR. We show that all of these are weak. We discuss our results, and conclude that generally, disguising is not a good idea.

Lots of people working in cryptography have no deep concern with real application issues. They are trying to discover things clever enough to write papers about.

Whitfield Diffie (1944–)

2 Cryptographic Background

We assume that the reader is familiar with the elementary concepts of cryptography. This chapter presents fundamental definitions and standard notation used throughout this thesis and may be safely skipped upon first reading.

For more detailed literature on the background of cryptography and its applications, the reader is recommended to consult the superb *Handbook of Applied Cryptography* [53] or other similar works: [25, 69, 77].

2.1 Complexity Theory

Complexity theory provides mechanisms for classifying computational problems according to the resources required to solve them, [53]. It is the goal of this theory that a problem may be described in terms of its fundamental difficulty and not one dependent on which computational model/machine one uses to attempt to solve it. These computational resources usually indicate the time (*time–complexity*) and storage space (*storage–complexity*) required for a solution.

To present this theory fully, one must describe problems in terms of Turing machines or other equivalent abstracta. We simplify this however, and consider complexity in terms of algorithms running on a modern computer. For the full theory, one should consult the seminal [49] and [50].

2.1.1 Complexity Notation

In this thesis we will use the standard definitions and notation from [53]:

Definition 2.1.1. The *input size* is the total number of bits required to represent an instance of a problem.

Definition 2.1.2. The *running time* of an algorithm is the number of primitive operations (similarly 'steps') executed. A step can be a bit–operation, modular multiplication, curve addition et cetera.

One considers the input size to a problem as it allows us to see how the running time of an algorithm is effected when the size of the problem is changed. It is usually difficult to state or derive an exact running time for an algorithm. So one often uses *asymptotic* run–times which give an approximation as the input size increases.

Definition 2.1.3 (Order Notation)**.** Let f and g be two functions $f, g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. Then:

- (Asymptotic Upper Bound) $f(n) = O\big(g(n)\big)$ if there exists $c \in \mathbb{R}_{>0}$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$ where $n_0 \in \mathbb{N}$.
- (Asymptotic Lower Bound) $f(n) = \Omega\big(g(n)\big)$ if there exists $c \in \mathbb{R}_{>0}$ such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$ where $n_0 \in \mathbb{N}$.

Thus one can think of asymptotic notation as giving the running time of an algorithm with all constants removed. We will use both the asymptotic and non–asymptotic notation where appropriate in this thesis.

2.1.2 Complexity Classes

Complexity classes classify how difficult a problem is asymptotically:

Definition 2.1.4. A *constant–time algorithm* is one with a worst–case running time of $O(1)$.

Definition 2.1.5. A *linear–time algorithm* is one with a worst–case running time of $O(n)$ where n is the input size.

Definition 2.1.6. A *polynomial–time algorithm* is one with a worst–case running time of $O(n^c)$ where n is the input size and $c > 1$ a real constant.

Definition 2.1.7. An *exponential–time algorithm* is one with a worst–case running time of $O(c^n)$ where n is the input size and $c > 1$ a real constant.

Definition 2.1.8. A *subexponential–time algorithm* is one whose worst–case running time is a function lying between that of polynomial and exponential.

Clearly one has the relative growth rates of: $O(1) < O(n) < O(n^c) < O(c^n)$. We consider polynomial–time or faster algorithms as efficient and asymptotically slower algorithms as inefficient. However the degree of the polynomial in a polynomial–time algorithm is important, for this distinction only holds asymptotically: For a small problem, such as factoring a small integer, exponential methods such as trial–division will often prove faster than sub–exponential ones.

2.2 Asymmetric Cryptography

This thesis is concerned with *asymmetric cryptography* (equivalently *public–key cryptography*). Asymmetric cryptography was first[1] suggested in 1976 by Diffie & Hellman [20] to overcome the key–distribution problem (see §2.5.1) and as a method for authenticating an unknown party.

Its principal advantage is unlike symmetric cryptography, no pre–shared key is required to be established. Instead, the system comprises of a *private* and *public–key* pair which are mathematically linked under a *one–way function* (equivalently a *trapdoor function*).

This quasi–invertible map is one such that computation one way is trivial (encryption), but the inverse operation (decryption) is 'hard' without some additional information; namely the private–key. This fundamental construction forms the basis of all deployed asymmetric cryptosystems.

[1] It was later declassified that J. Ellis of GCHQ discovered an equivalent definition of asymmetric cryptography in 1969 although this remained classified until the 1990's.

2.2.1 Computational Security and Hard Problems

We are interested in what constitutes a difficult computational problem in the 'real world' and as an example, we consider factoring large integers. Using the special number field sieve (SNFS), Aoki et al. in 2007 announced the factorisation of the 1039^{th} Mersenne prime $2^{1039} - 1$, [3, 4]. Here, a 22–bit factor was already known and the remaining two factors were unbalanced (265 and 752–bits respectively) so this was equivalent to factoring a general 700–bit integer. In total the team spent 127.5 AMD Opteron 2.2GHz years (and 127.5×4Gb of space) to complete the computation. Naïvely this required between 2^{68} and 2^{70} bit operations, depending on the balance between 32 and 64–bit code size and whether both processor cores were fully utilized.[2] To date, this is the largest computational problem known to have been solved.

In this thesis we will consider a computational problem as *tractable* if it can be solved in less than 2^{70} bit operations. Similarly a computational problem is *hard* if it requires significantly more than this effort (e.g. 2^{80} bit operations).

All known asymmetric systems are conjectured to be hard and their construction becomes a dynamic arms–race: Since the strength of a given cryptosystem is dependent on the key–size employed, when an improved attack or a significant computational advance is discovered, key–sizes must be increased accordingly.

2.3 The Discrete Logarithm Problem

The *discrete logarithm problem* or more commonly DLP, forms the corner–stone of our research here and so is presented in detail. Formally:

Definition 2.3.1. Let $(G, \oplus)$ be a finite cyclic group of prime order ℓ and let P be a generator of G. The (additive) map

$$\begin{aligned} \varphi : \mathbb{Z}_\ell &\to G \\ t &\mapsto [t]P = \underbrace{P \oplus P \oplus \cdots \oplus P}_{t-\text{times}} \end{aligned}$$

is an isomorphism between $(\mathbb{Z}_\ell, +)$ and $(G, \oplus)$. The problem of computing φ^{-1} is called the *discrete logarithm problem* to the base P. Specifically; given $P, Q \in \langle P \rangle$ determine the $t \in \mathbb{N}$ such that $Q = [t]P$, i.e: Find $t = \log_P Q$.

Up to isomorphism there is only one group $(G, \oplus)$, however it is well known by cryptographers that different representations of $(G, \oplus)$ can give different computational abilities to a cryptanalyst. For example, when $(G, \oplus)$ is the additive group of integers modulo ℓ, $(\mathbb{Z}_\ell, +)$, the DLP is reduced to computing a $t \in \mathbb{N}$ such that $ta = b \pmod{\ell}$ for a given $a, b \in \mathbb{Z}_\ell$. This is trivial as it only requires us to calculate a modular inverse which can be computed in $O(\lg^2 \ell)$ bit operations (polynomial–time) using the extended Euclidean algorithm [39]. Instead however; if we let $(G, \oplus) = E(\mathbb{F}_q)$ be the group of rational points on an

[2] Since the computational complexity for the SNFS to factor an integer n is

$$O\big(e^{(\frac{32}{9}\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}}\big)$$

this was equivalent to approximately 68–bits of work, [44]. Hence this result is reasonable.

elliptic curve defined over a finite field $\mathbb{F}_q$, generically the best known algorithm for solving this DLP is fully exponential in $\lceil \lg q \rceil$ [9].

It is thus clear that both the *specific* representation of $(G, \oplus)$ and the size of this group affect its suitability to be used as a cryptographic primitive.

2.4 Attacks on the DLP: Pohlig–Hellman

Let $(G, \oplus)$ be a finite cyclic group of (not necessarily prime) order $n > 1$. That is; G is generated by a single element $a \in G$ such that $\langle a \rangle = G$. Let $e \in G$ be the *neutral* element of G such that $a + e = a$ for all $a \in G$. We say the *order* of an element $a \in G$ is the integer $k \geq 1$ such that $[k]a = e$. The fundamental theorem for cyclic groups (Propositions 4.1 and 4.2 of [42]) states:

Proposition 2.4.1. *Let G be a finite cyclic group of order $n > 1$. Then for all $a \neq e \in G$, the order of a divides n. Then every subgroup of G is also cyclic and there exists at most one subgroup of order $k \leq n$. If f is a homomorphism of G, then the image of f is cyclic also.*

Assume that the order of G is composite: $n = \prod p_i^{e_i}$. Then G is isomorphic to

$$\Theta : G \longrightarrow C_{{p_1}^{e_1}} \times \cdots C_{{p_t}^{e_t}}$$

where C_{p^e} is the cyclic group of order p^e, ([42], p95). The projection of Θ onto a component C_{p^e} is given by $\theta_p : G \to C_{p^e}$ where

$$\theta_p : a \mapsto [n/p^e]a.$$

Assume one has a DLP on G: Given an $a, b \in \langle a \rangle = G$ find an $x \in \mathbb{N}$ such that $b = [x]a$. Since θ_p is a group homomorphism, one has $\theta_p(b) = \theta_p([x]a)$, i.e:

$$[n/p^e]b = [n/p^e]([x]a)$$

in C_{p^e}. Hence, if one could compute x modulo all the $p_i^{e_i}$ for all i, one could use the Chinese Remainder Theorem to recover x, ([42], p94).

Thus the problem of computing the discrete logarithm is only as hard as computing it in the largest prime order subgroup of G. This fact is attributed to Pohlig–Hellman [61] and is the reason one assumes one works in this largest prime order subgroup.

2.5 Cryptographic Schemes Based on the DLP

Here we present two cryptographic schemes based on the discrete logarithm problem; the Diffie–Hellman key–agreement protocol [20] and the ElGamal cryptosystem [22]. This presentation is made for an arbitrary finite abelian group $(G, \oplus)$ where we assume we work in the maximal prime order subgroup $\langle P \rangle$ of order ℓ.

2.5.1 DHKA–Protocol

The Diffie–Hellman key–agreement protocol (DHKA) was introduced to solve the *key distribution problem*. Most encryption is done by symmetric cryptosystems due to their inherent speed, however the shared symmetric key needs to be

established before communication can commence. The DH–protocol is a way of agreeing a key between two parties, here by convention denoted Alice and Bob.

Parameter Initialisation: Let $P \in G$ have prime order ℓ. The system parameters $(G, \oplus, P)$ are then published.

Message Exchange: Alice picks a random $a \in [1, \ell - 1] \subseteq \mathbb{N}$ and computes the element $A = [a]P$. Alice sets her private–key to be $a \in \mathbb{N}$ and sends the message A to Bob. Bob analogously picks a random $b \in [1, \ell - 1] \subseteq \mathbb{N}$, computes $B = [b]P$, sets his private–key to be $b \in \mathbb{N}$ and sends B to Alice.

Key Derivation: Upon message exchange, Alice computes

$$R_A = [a]B = [a]([b]P) = [ab]P,$$

and Bob

$$R_B = [b]A = [b]([a]P) = [ba]P.$$

Since $R_A = R_B$, this element can be input into some *key–derivation function* to create a shared key for symmetric cryptography between Alice and Bob.

Since we assume the DLP is hard in $(G, \oplus)$, an eavesdropper Eve cannot recover the a from $A = [a]P$ and so Alice and Bob have successfully agreed a key over an insecure channel. Note that this scheme is susceptible to the *man–in–the–middle attack*. This is where Eve intercepts the message A being sent to Bob and replaces it with an $E = [e]P$ known to Eve. Now Bob has unknowingly computed a shared key with Eve and not Alice. Eve can now decipher any ciphertext sent by Bob intended for Alice. This attack is easily overcome in practice by using authentication methods such as PKI; see p361 & p492 of [53].

2.5.2 ElGamal–Protocol

ElGamal is a hybrid public–key cryptosystem based on the DHKA from §2.5.1, [22]. Let $\varphi : G \to \{0,1\}^n$ be some key–derivation function. Then;

Parameter Initialisation: Alice selects an element $P \in G$ of order ℓ. With a random $a \in [1, \ell - 1] \subseteq \mathbb{N}$ she then computes her public/private–key pair $(A = [a]P, a)$. The system parameters $(G, +, \varphi, P, A)$ are then published.

Encryption: Bob wishes to send the message $m \in \{0,1\}^n$ to Alice. He picks a random $b \in [1, \ell - 1] \subseteq \mathbb{N}$ and computes $B = [b]P$ and the ciphertext $C = m \oplus \varphi([b]A)$. He then sends (B, C) to Alice.

Decryption: Alice receives the pair (B, C) from Bob. She then computes

$$[a]B = [a]([b]P) = [b]([a]P) = [b]A$$

and uses this to evaluate

$$C \ominus \varphi([b]A) = m.$$

The randomised ephemeral key (or *nonce*) $b \in \mathbb{N}$ means that ElGamal is an example of *randomised encryption*.

Map from one potato to the other potato
. . . think about potatoes
Erdős Pál (1913–1996)

3

Mathematical Background: Finite Fields

Since the focus of this thesis is applications in cryptography, most of our work requires finite fields. A good background to this material is the excellent *Introduction to Finite Fields* by H. Lidl & H. Niederreiter [48], S. Lang's *Algebra* [42] and Z. Wan's *Lectures on Finite Fields and Galois Rings* [85].

3.1 The Structure of Finite Fields

Definition 3.1.1. A *finite field* or *Galois field* is a field K of finite cardinality.

Definition 3.1.2. In a field the order of the multiplicative neutral element e in the additive subgroup is called the *characteristic* of the field. We denote this via

$$\mathrm{char}(\mathbb{F}_q) = p.$$

It is well known that the characteristic of *any* field is either 0 or a prime number p ([42], p90). If $\mathbb{F}_q$ were to have characteristic 0 it would contain $\mathbb{Q}$ and be infinite, hence the characteristic of a finite field must be a prime p.

Definition 3.1.3. If p is a prime we denote by $\mathbb{F}_p$ the field $\mathbb{Z}/p\mathbb{Z}$ of p elements. We call this a *prime field*. If $q = p^d$ where $d > 1$ we denote the field of q elements by $\mathbb{F}_q$. We call this an *extension field*.

Theorem 3.1.4. ([85], p122) *For any prime p and natural number d, there exists a finite field $\mathbb{F}_{p^d}$ which is unique up to isomorphism.*

Lemma 3.1.5. *Let $\mathbb{F}_{q^n}$ be a finite field. Then there exists a proper subfield*

$$\mathbb{F}_{q^d} \subseteq \mathbb{F}_{q^n} \qquad \textit{if and only if} \qquad d|n$$

The smallest proper subfield of $\mathbb{F}_q$ is its prime field $\mathbb{F}_p$ where $\mathrm{char}(\mathbb{F}_q) = p$.

Definition 3.1.6. The *group of units* or *multiplicative subgroup* of $\mathbb{F}_q$ is denoted via $\mathbb{F}_q^*$. Clearly since $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}$, the order of this group is

$$\#\mathbb{F}_q^* = q - 1.$$

Lemma 3.1.7. *Let $\mathbb{F}_q$ be a finite field. Then the multiplicative subgroup $\mathbb{F}_q^*$ is cyclic; that is there exists an element $\gamma \in \mathbb{F}_q^*$ which generates $\mathbb{F}_q^* = \langle\gamma\rangle$. Such an element is called a* primitive element *of $\mathbb{F}_q^*$.*

From Definition 3.1.6 and Lemma 3.1.7 one has:

Corollary 3.1.8. *For any finite field $\mathbb{F}_q$ there are $\varphi(q-1)$ primitive elements where φ is the Euler Totient function*

$$\varphi(N) = \#\{x : 1 \leq x \leq N, \gcd(x, N) = 1\}.$$

Moreover, if $e|(q-1)$ then there exist exactly $\varphi(e)$ elements of order e in $\mathbb{F}_q^*$.

Hence the map $f(x) = x^e$ is a bijection if and only if $\gcd(e, q-1) = 1$. This leads us to two important definitions:

Definition 3.1.9. There exists a unique up to isomorphism (infinite) algebraic extension of $\mathbb{F}_q$ in which every polynomial $m(X) \in \mathbb{F}_q[X]$ splits completely; that is $m(X)$ factors to give the product of linear monomials

$$m(X) = \prod (X - x_i).$$

We call this field the *algebraic closure* of $\mathbb{F}_q$ and denote it by $\overline{\mathbb{F}}_q$.

Definition 3.1.10. The *q^{th}–power Frobenius automorphism* is the map

$$\begin{aligned} \sigma : \overline{\mathbb{F}}_q &\to \overline{\mathbb{F}}_q \\ \alpha &\mapsto \alpha^q. \end{aligned}$$

Definition 3.1.11. The *Galois group of L over K*, denoted $\mathrm{Gal}(L/K)$ is the group of K–automorphisms of L.

Theorem 3.1.12. *Every finite extension $\mathbb{F}_{q^n}/\mathbb{F}_q$ is Galois and the group*

$$\mathrm{Gal}(\mathbb{F}_{q^n}/\mathbb{F}_q) = \langle \sigma \rangle$$

is cyclic of degree n.

Using Lemma 3.1.5 and Definition 3.1.10, one has a simple membership test:

Lemma 3.1.13. *For a given $n \geq 1$, all proper subfields of $\mathbb{F}_{q^n}$ correspond to the divisors $d|n$. An element $\alpha \in \mathbb{F}_{q^n}$ belongs to a proper subfield $\mathbb{F}_{q^d}$ of $\mathbb{F}_{q^n}$ if and only if $\sigma^d(\alpha) = \alpha$.*

We now are able to define two important functions:

Definition 3.1.14. Let $\alpha \in \mathbb{F}_{q^n}$ and σ be the Frobenius map. Then

$$\mathrm{Tr}_{\mathbb{F}_{q^n}/\mathbb{F}_q}(\alpha) = \sum_{i=0}^{n-1} \sigma^i(\alpha) = \sum_{i=0}^{n-1} \alpha^{q^i} \tag{3.1.1}$$

is denoted the *trace* and

$$\mathrm{N}_{\mathbb{F}_{q^n}/\mathbb{F}_q}(\alpha) = \prod_{i=0}^{n-1} \sigma^i(\alpha) = \prod_{i=0}^{n-1} \alpha^{q^i} \tag{3.1.2}$$

is the *norm* of α over $\mathbb{F}_q$.

When $\mathbb{F}_{q^n}$ and $\mathbb{F}_q$ are clear from context, we just write $\mathrm{Tr}(\alpha)$ and $\mathrm{N}(\alpha)$ for $\mathrm{Tr}_{\mathbb{F}_{q^n}/\mathbb{F}_q}(\alpha)$ and $\mathrm{N}_{\mathbb{F}_{q^n}/\mathbb{F}_q}(\alpha)$ respectively with an $\alpha \in \mathbb{F}_{q^n}$.

Theorem 3.1.15. *For $\alpha, \beta \in \mathbb{F}_{q^n}$ and an $a \in \mathbb{F}_q$ one has*

(i)	$\mathrm{Tr}(\alpha) \in \mathbb{F}_q$;	(i')	$\mathrm{N}(\alpha) \in \mathbb{F}_q$;
(ii)	$\mathrm{Tr}(\alpha+\beta) = \mathrm{Tr}(\alpha) + \mathrm{Tr}(\beta)$;	(ii')	$\mathrm{N}(\alpha\beta) = \mathrm{N}(\alpha)\,\mathrm{N}(\beta)$;
(iii)	$\mathrm{Tr}(a\alpha) = a\,\mathrm{Tr}(\alpha)$ *hence,* *in particular,* $\mathrm{Tr}(a) = na$;	(iii')	$\mathrm{N}(a\alpha) = a^n\,\mathrm{N}(\alpha)$ *hence,* *in particular,* $\mathrm{N}(a) = a^n$;
(iv)	$\mathrm{Tr}(\alpha^q) = \mathrm{Tr}(\alpha)$;	(iv')	$\mathrm{N}(\alpha^q) = \mathrm{N}(\alpha)$.

Here (ii) *and* (iii) *show that the trace defines an $\mathbb{F}_{q^n}$–linear map.*

Proof. See ([85], p149). □

3.2 Representing Finite Fields

As indicated in Definition 3.1.3, we use the canonical set of representatives

$$\{0, 1, 2, \ldots, p-1\}$$

for prime fields, since $\mathbb{Z}/p\mathbb{Z} \cong \mathbb{F}_p$ where all arithmetic is done modulo p.

For extension fields however there are many equivalent ways in which their elements can be represented. Since $\mathbb{F}_{q^n}$ is a vector space over $\mathbb{F}_q$ it is natural to use a *basis* representation for $\mathbb{F}_{q^n}$. In this thesis we use both *polynomial* and *normal* representations for finite extension fields. As there exist efficient polynomial–time algorithms for the conversion of one representation to another, this is in no way restrictive. Much of this follows [16] where further details may be sought if needed:

3.2.1 Polynomial Representation

Denote by $\mathbb{F}_q[X]$ the ring of polynomials over $\mathbb{F}_q$. If $m(X)$ is a monic irreducible polynomial, then $\langle m(X)\rangle$ is a maximal ideal of $\mathbb{F}_q[X]$ and hence $\mathbb{F}_q[X]/\langle m(X)\rangle$ is a field. If $\deg(m(X)) = n$ then $\mathbb{F}_q[X]/\langle m(X)\rangle$ has q^n elements and must be isomorphic to $\mathbb{F}_{q^n}$.

A vector space basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ is $\{1, X, \ldots, X^{n-1}\}$. This basis is called a *polynomial basis for $\mathbb{F}_{q^n}$ with respect to $\mathbb{F}_q$*. Under this, all elements $\alpha \in \mathbb{F}_{q^n}$ may be written as

$$\alpha = \alpha_{n-1}X^{n-1} + \alpha_{n-2}X^{n-2} + \cdots + \alpha_1 X + \alpha_0$$

where all the $\alpha_i \in \mathbb{F}_q$.

Addition, subtraction and multiplication are carried out modulo $m(X)$ and $\mathrm{char}(\mathbb{F}_q)$ when using this representation. Inversion can be computed using the *extended Euclidean algorithm* over $\mathbb{F}_q[X]$, for details see von zur Gathen & Gerhard ([83], p46) or Knuth ([39], p342).

There are approximately q^n/n monic irreducible polynomials of degree n over $\mathbb{F}_q$, thus such a representation is clearly non–unique: Given two irreducible polynomials $m(X)$ and $g(Y)$ of degree n one has

$$\mathbb{F}_q[X]/\langle m(X)\rangle \cong \mathbb{F}_q[Y]/\langle g(Y)\rangle.$$

A natural problem is to explicitly compute the isomorphism between them. This can be done in polynomial–time and will be the subject of §3.3.

Any monic irreducible polynomial of suitable degree can be used to construct $\mathbb{F}_{q^n}$, but in reality one often uses those with special properties which help accelerate field arithmetic. Two properties one could look for are:

- The polynomial $m(X)$ is *primitive*. That is one of its roots is a primitive element, thus also a generator of the multiplicative subgroup $\mathbb{F}_{q^n}^*$. There are approximately $\varphi(q^n-1)/n$ monic primitive polynomials of degree n in $\mathbb{F}_q[X]$, hence these are plentiful in practice.
- The polynomial $m(X)$ is *sparse* (that is it contains very few non–zero coefficients). Such sparse polynomials allow for a faster reduction using known algorithms and are often recommended in practice. See [84].

Occasionally, one can find a polynomial which is both sparse and primitive: $X^{167}+X^6+1$ is such an example over $\mathbb{F}_2[X]$.

3.2.2 Normal Representation

Definition 3.2.1. [16] An element $\alpha \in \mathbb{F}_{q^n}$ is said to be *normal over* $\mathbb{F}_q$ if

$$\alpha, \sigma(\alpha), \sigma^2(\alpha), \ldots, \sigma^{n-1}(\alpha)$$

are linearly independent over $\mathbb{F}_q$. In this case, one can construct a basis of the vector space $\mathbb{F}_{q^n}/\mathbb{F}_q$ as

$$\{\alpha, \alpha^q, \alpha^{q^2}, \ldots, \alpha^{q^{n-1}}\}$$

which is denoted the *normal basis of* $\mathbb{F}_{q^n}$ *over* $\mathbb{F}_q$.

Hensel proved there always exists a normal basis for $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ for all n, and an element α is normal if and only if

$$\gcd\Big(\sum_{j=0}^{k-1} \sigma^{k-1-j}(\alpha) X^j, X^k - 1\Big) = 1.$$

For any $\beta \in \mathbb{F}_{q^n}$ in this representation, the computation $\sigma(\beta) = \beta^q$ is very easy since it is just a cyclic shift of the co–ordinates of β with respect to the fixed basis $\{\alpha, \sigma(\alpha), \sigma^2(\alpha), \ldots, \sigma^{n-1}(\alpha)\}$. This is especially useful when representing $\mathbb{F}_{2^n}$ since it becomes just a bit–shift which is especially cheap to do in hardware.

3.3 Computing Isomorphisms Between Finite Fields

For every prime power p^n there exists a finite field $\mathbb{F}_{p^n}$ which is unique up to isomorphism. If f and g are two monic irreducible polynomials over $\mathbb{F}_p$ of degree n, then the quotient rings $K_f := \mathbb{F}_p[X]/\langle f(X)\rangle$ and $K_g := \mathbb{F}_p[Y]/\langle g(Y)\rangle$ represent isomorphic finite fields. Here K_f has elements expressed as polynomials $\alpha_{n-1}X^{n-1} + \alpha_{n-2}X^{n-2} + \cdots + \alpha_0$ modulo $(g(X), p)$ and likewise K_f has elements $\beta_{n-1}Y^{n-1} + \beta_{n-2}Y^{n-2} + \cdots + \beta_0 \mod (f(Y), p)$. The aim here is to compute the explicit isomorphism which maps representations in K_f to that of those in K_g.

All methods for computing an isomorphism between finite fields rely on finding a certain element in one field, and then decomposing its representation in terms of the basis of the other. A map can then be constructed in terms of these basis elements.

Pinch in [60] gave two algorithms for computing such an isomorphism. The idea of these methods is work in some group Γ defined by algebraic operations over $\mathbb{F}_p$. Then to pick a $\gamma \in \mathbb{F}_{p^n}$ of small order r and simultaneously represent γ as an element $\alpha_x \in K_f$ in the root x of f and as an element $\beta_y \in K_g$ in the root y of g. If γ generates $\mathbb{F}_{p^n}^*/\mathbb{F}_p$ and no other proper subgroup $\mathbb{F}_{p^d}^*$ of $\mathbb{F}_{p^n}^*$, then one can find a relation since both α_x and β_y represent an element of small order r, and so one must have $\phi(\alpha_x) = \beta_y^s$ for some $1 \leq s \leq r$ and an isomorphism ϕ. One can then use this to express the defining polynomial of K_f in terms of y. This method was not efficient for large fields however since one may need to test r cases for an isomorphism between $\phi(\alpha_x) = \beta_y^s$.

Better polynomial–time methods do exist, and here we present this idea using the simplest (but not efficient) algorithm:

Algorithm 3.3.1 (Polynomial–time algorithm for the computation of a Finite Field Isomorphism)**.**
INPUT: Fields $\mathbb{F}_p[X]/\langle f(X)\rangle$ and $\mathbb{F}_p[Y]/\langle g(Y)\rangle$ such that $\deg f = \deg g$;
OUTPUT: A rational map m such that $f \circ m \equiv 0 \pmod{g}$.

1. Assume that $f(X)$ is square–free (no repeated roots);
2. Repeatedly use Berlekamp's/Cantor–Zassenhaus algorithm to factorise $f(Y) \in \mathbb{F}_p[Y]/\langle g(Y)\rangle$ into irreducible factors;
3. Set the simplest root of $f(Y)$ as θ;
4. Now use this root to define the map

$$\begin{aligned} \mathbb{F}_p[X]/\langle f(X)\rangle &\to \mathbb{F}_p[Y]/\langle g(Y)\rangle, \\ X &\mapsto \theta. \end{aligned}$$

This algorithm is polynomial–time but far from optimal. Lenstra in [47] was the first to show that polynomial–time algorithms do exist, but gave no explicit algorithms for constructing a polynomial $m \in K_g$ such that $f \circ m \equiv 0 \pmod{g}$. The first implementable polynomial–time method was given by Allombert in [1] and the interested reader should consult this paper for details.

Part I
Compression

Knowledge in youth is wisdom in age.
Anon (unknown)

4 Background

4.1 Elliptic Curves

Here we present the bare minimum of results required for our study of cryptography. There have been hundreds of years of research on elliptic and higher genus curves and this is reflected by the wealth of material that is available. For the underlying theory our research references the excellent books by Silverman [74, 75] and Washington [86]. For implementation issues of ECC, we utilise the books by Hankerson, Menezes & Vanstone [37], Enge [23], *The Handbook of Elliptic and Hyperelliptic Curve Cryptography* [16] and the excellent duo; *Elliptic Curves in Cryptography* and *Advances in Elliptic Curve Cryptography* [9, 78]. We present the main results we require now:

Definition 4.1.1. An elliptic curve E defined over a finite field $\mathbb{F}_q$ is given by the affine *Weierstraß* equation

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

where the $a_i \in \mathbb{F}_q$ and E has a non–zero discriminant. For simplicity one usually writes $E/\mathbb{F}_q$ when the curve E is defined over the field $\mathbb{F}_q$.

When one is working over finite fields with characteristic coprime to 6, one may define an elliptic curve via the simpler short–Weierstraß form:

Definition 4.1.2. Let E be an elliptic curve defined over a finite field whose characteristic p is not 2 nor 3. Then one can define E via the *short–Weierstraß* form

$$y^2 = x^3 + ax + b$$

where $a, b \in \mathbb{F}_q$ and the discriminant $4a^2 + 27b^2 \not\equiv 0 \pmod{p}$.

Definition 4.1.3. The set of points on any elliptic curve $E/\mathbb{F}_q$ is denoted

$$E(\mathbb{F}_q) := \{(x, y) \in \mathbb{F}_q \times \mathbb{F}_q : y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\},$$

where the projective element $\mathcal{O}$ is called the *point at infinity*.

Theorem 4.1.4. *The set $E(\mathbb{F}_q)$ forms an additive abelian group with identity $\mathcal{O}$ under* point addition*, see §4.1.1.*

Since $\mathbb{F}_q$ is finite, $E(\mathbb{F}_q)$ is necessarily a finite abelian group and may be used for DL–based cryptosystems. This was first suggested simultaneously by Koblitz [40] and Miller [55] in 1985.

Definition 4.1.5. Let $E/\mathbb{F}_q$. Then the q^{th}*–power Frobenius map* is

$$\psi : \begin{cases} E(\overline{\mathbb{F}_q}) & \to \ E(\overline{\mathbb{F}_q}) \\ (x,y) & \mapsto (x^q, y^q), \\ \mathcal{O} & \mapsto \ \mathcal{O}. \end{cases}$$

The map ψ is a group endomorphism of $E(\mathbb{F}_{q^n})$ for any $n \geq 1$ since it respects the elliptic curve group–law; $\psi(P+Q) = \psi(P) + \psi(Q)$.

Definition 4.1.6. Let $E/\mathbb{F}_q$. Then the quantity

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

is called the *trace of Frobenius at* q.

Theorem 4.1.7. *The map* ψ *satisfies the relation*

$$\psi^2 - [t]\psi + [q] = [0],$$

called the characteristic equation of ψ. *That is, for any point* $P = (x,y) \in E(\mathbb{F}_q)$ *one has*

$$(x^{q^2}, y^{q^2}) - [t](x^q, y^q) + [q](x,y) = \mathcal{O}.$$

To use the group $E(\mathbb{F}_q)$ in cryptography, we need to know its order. The following theorem by Haße gives us a bound for the size of this group:

Theorem 4.1.8. *Let* $E/\mathbb{F}_q$. *Then the trace of Frobenius satisfies*

$$|t| \leq 2\sqrt{q}.$$

The set of integers $\{x \in \mathbb{N} : q + 1 - 2\sqrt{q} \leq x \leq q + 1 + 2\sqrt{q}\}$ *is called the* Haße–Interval.

In the sequel we will need to be able to describe certain properties of elliptic curves E. We give definitions for these now:

Definition 4.1.9. Let $E/\mathbb{F}_q$ and $r \in \mathbb{N}$. Then

$$E[r] := \{P \in E(\overline{\mathbb{F}}_q) : [r]P = \mathcal{O}\}$$

is the set of r*–torsion points* of E. That is the set of points of order r.

Theorem 4.1.10. *Let* $E/\mathbb{F}_q$ *and* $\mathrm{char}(\mathbb{F}_q) = p$. *Then if* r *is coprime to* p,

$$E[r] \cong \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}.$$

Otherwise, $r = p^s$ *and either*

$$E[p^s] = \{\mathcal{O}\} \qquad or \qquad E[p^s] \cong \mathbb{Z}/p^s\mathbb{Z}$$

for all $s \geq 1$.

Definition 4.1.11. Let $E/\mathbb{F}_q$ and $\mathrm{char}(\mathbb{F}_q) = p$. If $E[p^s] = \{\mathcal{O}\}$ for all $s \geq 1$, then E is said to be *supersingular*. Otherwise it is called *ordinary*.

Finally, one defines the extraction functions required in the sequel:

Definition 4.1.12. Let $\mathrm{x}(P) : E(\mathbb{F}_q) \to \mathbb{F}_q$ be the function which extracts the abscissa and $\mathrm{y}(P) : E(\mathbb{F}_q) \to \mathbb{F}_q$ be one which extracts the ordinal of $P \in E(\mathbb{F}_q)$.

4.1.1 Addition Law for Curve Points

The points of $E(\mathbb{F}_q)$ are well known to form a group under elliptic curve point addition, as Theorem 4.1.4 stated. In this thesis we require formulæ solely for the binary fields case. We present such formulæ here. For the general case one should consult [75].

For Curves over Binary Fields

For the elliptic curve $E/\mathbb{F}_{2^n}$ defined by the short–Weierstraß form

$$y^2 + xy = x^3 + ax^2 + b$$

one has the following formulæ [9]:

For the points $P_i = (x_i, y_i) \in E(\mathbb{F}_{2^n})$ and the neutral element $\mathcal{O}$ one has:

$$-P_i = (x_i, x_i + y_i).$$

To compute $P_3 \leftarrow P_1 + P_2$ **do**

`A:` **if** $P_2 = -P_1$ **then:**

- **set** $P_3 \leftarrow \mathcal{O}$.
- **goto step** `C`.

else if $(x_1 \neq x_2)$, **set**

$$\lambda \leftarrow \frac{y_2 + y_1}{x_2 + x_1}, \qquad \mu \leftarrow \frac{y_1 x_2 + y_2 x_1}{x_2 + x_1}.$$

else if $(x_1 = x_2 \neq 0)$, **set**

$$\lambda \leftarrow \frac{x_1^2 + y_1}{x_1}, \qquad \mu \leftarrow x_1^2.$$

`B:` **set** $P_3 \leftarrow (x_3, y_3)$ **where**

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + a + x_1 + x_2, \\ y_3 &= (\lambda + 1)x_3 + \mu \\ &= (x_1 + x_3)\lambda + x_3 + y_1. \end{aligned}$$

`C:` **return** P_3. **halt**.

Lemma 4.1.13. *In an odd prime order subgroup ℓ of $E(\mathbb{F}_{2^n})$, all points $P = (x_2, y_2) \neq \mathcal{O}$ may be written as $P = [2]R$ where $R = (x_1, y_1)$ and*

$$x_2 = x_1^2 + b x_1^{-2}.$$

Proof. If we are working in a prime order subgroup, for an $R \in \langle P \rangle$ one has $\exists b \in [1, \ell]$ such that $P = [b]R$. Since ℓ is odd, one can find an $R \in \langle P \rangle$ such that $P = [2]R$.

Let $P = (x_2, y_2)$ and $R = (x_1, y_1)$. From the formulæ above one has that

$$x_2 = \lambda(\lambda + 1) + a, \qquad \lambda = (x_1^2 + y_1)x_1^{-1}.$$

Substituting in for λ gives

$$\frac{x_1^4 + y_1^2}{x_1^2} + \frac{x_1^2 + y_1}{x_1} + \frac{ax_1^2}{x_1^2} = \frac{x_1^4 + x_1^3 + a_2x_1^2 + (y_1^2 + x_1y_1)}{x_1^2}$$

where with $y_1^2 + x_1y_1 = x_1^3 + ax_1^2 + b$ the result subsequently follows. □

4.1.2 Point Compression

An elliptic curve point $P = (x, y) \in E(\mathbb{F}_q)$ where $n = \lceil \lg q \rceil$ naïvely requires $2n$–bits to store or transmit; n–bits for each $\mathbb{F}_q$ element. However, for each abscissa x there exists at most two corresponding ordinals y and y' corresponding to the roots of the Weierstraß equation [9]. Hence one can send x and a bit b which determines the choice of ordinal. The receiver can then uniquely reconstruct the point by solving a quadratic equation. This method only requires $(n + 1)$–bits to specify P and is denoted *point compression*. The methods for such are:

For curves over non–binary fields

For a curve $E/\mathbb{F}_q$ where $\mathbb{F}_q = \mathbb{F}_{p^n} = \mathbb{F}_p(\theta)$ one has $-P = (x, -y)$ if $P = (x, y)$, with $P = -P$ if and only if $y = 0$. Since $-y = q - y$ and q is odd, the parity of y and $-y$ are opposite unless $y = 0$.

Thus one sends $(x, b(y))$ to represent $P = (x, y)$ where $b(y)$ is a bit equal to zero when c_0 in $y = \sum_{i=0}^{n-1} c_i\theta^i$ is even, and 1 otherwise.

The point is recovered by computing from the Weierstraß form an $\pm y$, with it's sign being determined by $b(y)$.

For curves over binary fields

For a curve $E/\mathbb{F}_{2^n} : y^2 + xy + x^3 + ax^2 + b = 0$, one has $-P = (x, x + y)$ if $P = (x, y)$, with $P = -P$ if and only if $y = 0$. First let us consider the following:

Theorem 4.1.14. ([85], p156) *Consider the quadratic equation*

$$z^2 + z + \beta = 0 \tag{4.1.1}$$

over $\mathbb{F}_{2^n}$ *where* $\beta \in \mathbb{F}_{2^n}$. *This has valid solutions* $z, z+1$ *if and only if* $\mathrm{Tr}(\beta) = 0$.

Dividing the Weierstraß equation $E : y^2 + xy = x^3 + ax^2 + b$ by x^2 and setting $z = y/x$ gives us

$$z^2 + z + \beta = 0$$

where $\beta = x + a + bx^{-2}$ which is known to have a valid solutions $z, z + 1$ if and only if $\mathrm{Tr}(\beta) = 0$ from Theorem 4.1.14. Hence, to send a point (x_1, y_1) one computes $z_1 = (y_1/x_1)$ and sends (x_1, c_0), where c_0 is the least significant bit of $(y_1/x_1) = \sum_{i=0}^{n-1} c_i 2^i$. If $y_1 = 0$ we set $c_0 = 0$ and only one point P with an $x_1 = 0$ exists, hence the parity of y_1 here is not required.

To recover y_1 for a $x_1 \neq 0$, one recovers a solution z_1 of

$$z^2 + z = x_1 + a + bx^{-2}.$$

If the least significant bit of z_1 equals c_0, we set $y_1 = x_1z_1$. Otherwise one sets $y_1 = x_1(z_1 + 1)$ and we are done.

4.1.3 Seroussi's Point Compression for Curves $E/\mathbb{F}_{2^n}$

For any elliptic curve $E/\mathbb{F}_{2^n}$, using *Seroussi's method* from [71] will save one bit in the representation of an abscissa of a point.

One knows from Theorem 3.1.15 that: $\mathrm{Tr}(\alpha) \in \mathbb{F}_q$, $\mathrm{Tr}(\alpha+\beta) = \mathrm{Tr}(\alpha)+\mathrm{Tr}(\beta)$ and $\mathrm{Tr}(\alpha^q) = \mathrm{Tr}(\alpha)$ for all $\alpha, \beta \in \mathbb{F}_{q^n}$. Under this one has:

Lemma 4.1.15. *Let $E/\mathbb{F}_{2^n}$ be defined by the Weierstraß equation*

$$E : y^2 + xy = x^3 + ax^2 + b$$

and $\langle P \rangle \subseteq E(\mathbb{F}_{2^n})$ be of odd prime order ℓ. Then for all points $Q \in \langle P \rangle$ one has

$$\mathrm{Tr}(\mathrm{x}(P)) = \mathrm{Tr}(a),$$

a binary constant.

Proof. From Theorem 4.1.14, dividing the Weierstraß equation $E : y^2 + xy = x^3 + ax^2 + b$ by x^2 and setting $z = y/x$ gives us $z^2 + z + \beta = 0$ where $\beta = x + a_2 + bx^{-2}$. This has solutions $z, z+1$ if and only if $\mathrm{Tr}(\beta) = 0$. From Lemma 4.1.13, in an odd prime order subgroup of $E(\mathbb{F}_{2^n})$, all points $R = (x_2, y_2) \neq \mathcal{O}$ may be written as $R = [2]P$ where $P = (x_1, y_1)$ and $x_2 = x_1^2 + bx_1^{-2}$. As P is a point on the curve it must satisfy (4.1.1) with $z = y/x$. Thus one has

$$\mathrm{Tr}\left(x_1 + a + \frac{b}{x_1^2}\right) = 0.$$

Using rules for the trace, one has $\mathrm{Tr}(\alpha^2) = \mathrm{Tr}(\alpha)$ and with $\alpha = -\alpha \in \mathbb{F}_{2^n}$ this gives

$$\mathrm{Tr}\left(x_1^2 + a + \frac{b}{x_1^2}\right) = 0 \Longleftrightarrow \mathrm{Tr}(a) = \mathrm{Tr}\left(x_1^2 + \frac{b}{x_1^2}\right) = \mathrm{Tr}(x_2).$$

Hence, for all points P in this odd prime subgroup $\mathrm{Tr}(\mathrm{x}(P)) = \mathrm{Tr}(a) \in \mathbb{F}_2$. □

As the trace is a linear operator ([42], p119), it can be computed as an inner product:

Theorem 4.1.16. *Let $\mathbf{x} = \sum_{i=0}^{n-1} x_i\alpha_i$ for some basis $\{\alpha_0, \ldots, \alpha_{n-1}\}$ of $\mathbb{F}_{2^n}/\mathbb{F}_2$ and $t_i = \mathrm{Tr}(\alpha_i)$. Then*

$$\mathrm{Tr}(x) = \overline{t} \cdot \overline{x} = \sum_{i=0}^{n-1} t_i x_i.$$

Let $\overline{t}$ be defined as above and let j be smallest index such that $t_j = 1$. Now when we specify the abscissa $\mathrm{x}(P) = \mathtt{x}_{n-1}\mathtt{x}_{n-2}\cdots\mathtt{x}_1\mathtt{x}_0$, we only transmit the punctured binary string

$$\mathtt{x}' = \mathtt{x}_{n-1}\mathtt{x}_{n-2}\cdots\mathtt{x}_{j+1}\mathtt{x}_{j-1}\cdots\mathtt{x}_1\mathtt{x}_0$$

which is of length $n-1$.

As we are working with the largest prime order subgroup, we have $\overline{t} \cdot \overline{x} = \mathrm{Tr}(a)$ and so

$$\mathtt{x}_j = \left(\mathrm{Tr}(a) + \sum_{i=0}^{j-1} t_i x_i + \sum_{i=j+1}^{n-1} t_i x_i\right) \pmod 2.$$

This allows one to recover the original abscissa x from the pair $(\mathtt{x}', a_2)$, since a posteriori the representation of $\mathbb{F}_{2^n}$ is known, and thus so is $\overline{t}$.

Lemma 4.1.17. *When using a polynomial basis of $\mathbb{F}_{2^n}/\mathbb{F}_2$, for an odd n and $x \in \mathbb{F}_{2^n}$ represented by the vector $\mathbf{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)$ one has*

$$\mathrm{Tr}(\mathbf{x}) = x_0,$$

the least significant bit of $\mathbf{x}$.

Lemma 4.1.18. *When using a normal basis of $\mathbb{F}_{2^n}/\mathbb{F}_2$, for an $x \in \mathbb{F}_{2^n}$ represented by the vector $\mathbf{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)$ one has*

$$\mathrm{Tr}(\mathbf{x}) = \sum_{i=0}^{n-1} x_i.$$

Proof. For a given $\mathbf{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)$ one has

$$\mathbf{x} = \sum_{i=0}^{n-1} x_i \beta^{2^i}, \sigma(\mathbf{x}) = \sum_{i=0}^{n-1} x_i \beta^{2^{i+1}} = \sum_{i=0}^{n-1} x_{i-1} \beta^{2^i}, \ldots, \sigma^j(\mathbf{x}) = \sum_{i=0}^{n-1} x_{i-j} \beta^{2^i}$$

as squaring in normal representation is merely a bit–shift. Hence

$$\mathrm{Tr}(\mathbf{x}) = \sum_{j=0}^{n-1} \sigma^j(\mathbf{x}) = \sum_{i=0}^{n-1} \Big(\sum_{j=0}^{n-1} x_{i-j} \Big) \beta^{2^i} = S \sum_{i=0}^{n-1} \beta^{2^i} = S \cdot \mathrm{Tr}(\beta)$$

where the index $(i-j)$ is modulo n and $S = \sum_{j=0}^{n-1} x_j$. It is well known that $\mathrm{Tr}(\beta) = 1$ if β generates a normal basis. This result is equivalently shown by the trace being basis independent. The result now follows. □

4.2 Koblitz Curves

N. Koblitz in [41] demonstrated the performance benefits of using the group of $\mathbb{F}_{2^n}$–points of non–supersingular curves over $\mathbb{F}_2$. These are recommended for use in the Elliptic Curve Digital Signature Standard (ECDSA) by NIST in [58].

Definition 4.2.1. [16] A *Koblitz curve* (equivalently an *anomalous binary curve*) is an elliptic curve $E/\mathbb{F}_2$ given by the Weierstraß equation

$$y^2 + xy = x^3 + ax^2 + 1 \qquad \text{with } a \in \{0, 1\}. \tag{4.2.1}$$

We consider the group of points $E(\mathbb{F}_{2^n})$ for cryptography.

Koblitz curves have two key advantages over general ones: Primarily, doublings can be replaced by computations involving the Frobenius automorphism which are much more efficient. To see an example of this, consider the characteristic equation of the Frobenius from Theorem 4.1.7:

$$\psi^2 - \mu\psi + 2$$

where $\mu = (-1)^{1-a_2}$. Thus for all $P \in E_{a_2}(\mathbb{F}_{2^n})$, one can replace doublings by

$$[2]P = [\mu]\psi(P) - \psi^2(P).$$

Scalar multiplication by powers of two is now very fast as we have replaced a point doubling by a point subtraction and evaluation of $\psi(P)$ from P.

More efficient methods for scalar multiplication using Frobenius expansions exist and the interested reader should consult the work of Solinas in [81, 82].

Secondarily, one is able to evaluate the lifted order of $\#E(\mathbb{F}_{2^n})$ very efficiently, using only arithmetic in $\mathbb{Z}$. See §5.6.1 for further details.

4.3 Edwards Curves

Rather excitingly in 2007, Edwards' in [21] gave an alternative form other than the Weierstraß equation to describe an elliptic curve. This was significant, as one can give a *strongly unified* addition law on the curve. This is where one formula describes point addition, point doubling and point inversion with no exceptional points. Apart from simplicity, an additional advantage of having a unified addition/doubling formula is resistance against *side–channel attacks*. This is unlike the standard Weierstraß form commonly used, and makes Edwards curves very interesting for cryptography.

Subsequently, Bernstein & Lange in [7, 8] showed that Edwards curves have exceptionally efficient arithmetic, particularly when using small curve coefficients. We now briefly present Edwards curves, and refer the interested reader to [21], and particularly [7] for more details.

Definition 4.3.1. Let K be a field not of characteristic 2. Fix a $c, d \in K$ where $cd(1 - dc^4) \neq 0$ and $d \in K - \{0, 1\}$ is non–square. Then an *Edwards curve over* K is one of the form

$$x^2 + y^2 = c^2(1 + dx^2y^2).$$

Theorem 4.3.2. *The set of points*

$$\{(x, y) \in K \times K : x^2 + y^2 = c^2(1 + dx^2y^2)\}$$

forms a commutative group with $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ *defined by the Edwards addition law*

$$x_3 = \frac{x_1y_2 + y_1x_2}{c(1 + dx_1x_2y_1y_2)}, \qquad y_3 = \frac{y_1y_2 + x_1x_2}{c(1 - dx_1x_2y_1y_2)}$$

with neutral element $(0, c)$ *and* $-(x, y) = (-x, y)$.

Originally, Edwards set $d = 1$ and showed that over $\overline{K}$, every Edwards curve is birationally equivalent to an elliptic curve in Weierstraß form defined over K. However, by extending the theory to allow more general $d \in K - \{0, 1\}$, Bernstein & Lange in [7] showed that just over a $1/4$ of all curves over a finite field of odd characteristic, can be transformed into Edwards form:

Theorem 4.3.3. ([7], p4) *Let* K *be a finite field in which* $2 \neq 0$. *Let* E/K *such that the group of points* $E(K)$ *has an element of order 4 and a unique element of order 2. Then there exists a non–square* $d \in K$ *such that the curve* $x^2 + y^2 = 1 + dx^2y^2$ *is birationally equivalent to* E *over* K.

Every Edwards curve has a point of order 4, namely $(c, 0)$. Hence when we consider mappings between the forms, it is natural to use elliptic curves having a point of order four.

In our work here we do not need to compute maps between Edwards and Weierstraß forms. Instead using the work of Bernstein & Lange in [7], we consider point addition being *directly* evaluated on the Edwards curve.

4.4 Security Parameters

The security of ECDLP–based cryptosystems is based on the size of the largest prime divisor ℓ of $\#E(\mathbb{F}_q)$ and the complexity of the best known attacks. For general curves, these are the exponential–time Pollard–ρ methods given by Wiener & Zuccherato [87] and Gallant, Lambert & Vanstone [32]. These results hold for all except a few known weaker curves (see ([9], p79–99) and [16]), which we shall assume are not admitted throughout the rest of our work here. This leads us to the following definition which is an analogue of those often used in literature:

Definition 4.4.1. Let the *security parameter* of a cryptographic system based on the ECDLP be defined as

$$k := \begin{cases} \lfloor \lg \ell \rfloor & \text{for general curves, } E/\mathbb{F}_q, \\ \lfloor \lg \ell - \lg n \rfloor & \text{for Koblitz curves, } E/\mathbb{F}_{2^n}. \end{cases}$$

This implies the best known attack requires an expected

$$2^{k/2}$$

steps to compute logarithms. One sets the value of k to be the floor of $\lg \ell$ to ensure that a system of order ℓ has *at least* a security parameter of k.

If A equals success, then the formula is

$$A = X + Y + Z.$$

X is work. Y is play. Z is keep your mouth shut.

Albert Einstein (1879–1955)

5 Compression of Elliptic Curve Domain Parameters

5.1 Motivation

The principal advantage of elliptic curve based cryptography is per–bit–security. However in the initialisation of public–key cryptosystems, so called *domain parameters* need to be established/transmitted which when naïvely implemented, require extensive bandwidth.

Typically for ECMQV, ECDSA [2, 13] and other standardised protocols the domain parameters include: the field, the Weierstraß coefficients of the elliptic curve, a point of prime order ℓ, this order ℓ and the cofactor of the curve.

Some elliptic curve based systems use fixed domain parameters, however this approach can be disadvantageous: It lacks flexibility to increase key–sizes with computational and algorithmic advances, it inhibits curve performance as one cannot tailor parameters to software or hardware, and more fundamentally perhaps; who is to select the parameters and who will trust them not to specify a (not publically known) weaker curve? Clearly for such a system, the parameters are never communicated and are hardwired into implementations.

An alternative system is one which allows its users to specify their own domain parameters. We will call the latter *multi–curve environments*. Here, we assume these system parameters will need to be communicated.

The motivation for our research is to reduce this bandwidth: In multi–curve environments this issue becomes more important as many parameters could be stored in some database. These parameters may subsequently be transmitted multiple times. Hence, a natural question to ask is: *Can one reduce the size of these parameters without effecting the security and relative efficiency of such systems?* This is the main problem that we consider here.

There are two approaches: In this chapter, we consider the general case by analysing the bandwidth required by SEC 1 for transmitting domain parameters. Subsequently we investigate whether this bandwidth can be reduced, without restricting the systems representable under SEC 1.

The second approach is to use an alternative model which only defines restricted families; of curves and fields. This will be the subject of Chapter 6.

There has been little research in this area: The only results we are aware of for the general case are Smart given in §5.3, and for the restricted case Smart in §6.2 and Brown, Myers & Solinas considered in §6.3. It is reasonable that non experts may implement ECDLP–based systems explicitly on what is presented

in such standards as [2, 13, 14]. We show that the descriptions in these standards are not optimal.

Constrained environments such as smart–cards which have a restricted allocation of memory benefit from this research. Methods which reduce the storage requirements for domain parameters but greatly increase the computational effort are explicitly highlighted, since for computationally restricted devices they will prove less beneficial.

5.2 Domain Parameters

Here we present a standard for specifying domain parameters when implementing asymmetric ECC. For this we use the latest working draft (version 1.7, November 2006) of the *Standards for Efficient Cryptography (SEC 1): Elliptic Curve Cryptography*, [15]. Throughout the sequel, this will be used as a guide with which to compare our results to.

SEC 1 explicitly describes how one should represent domain parameters for transmission and storage including: finite field elements, natural numbers and points on elliptic curves. This is principally done via octet strings, the details of which is presented in Appendix A.

Throughout this chapter, we will be analysing and comparing the specific bits required to store and transmit certain data types. For this we use the following notation:

Definition 5.2.1. Let x be an element of arbitrary type (e.g. an octet–string, a finite field element, an elliptic curve point, et cetera) stored digitally using some data–type (e.g. an octet–string). Then the expected amount of bits required to represent an arbitrary element using a specific data–type is denoted as its *bit–size*, and will be expressed by

$$\|x\| .$$

Usually when one defines domain parameters an additional value $Q \in E(\mathbb{F}_q)$ is included, which is a user's public–key. In general the best one can do is via point–compression from §4.1.2. However for the Koblitz curve case over $\mathbb{F}_{2^n}$ one can actually do better than previously published. Methods for this are discussed later in Chapter 7.

5.2.1 SEC 1: Elliptic Curve Domain Parameters

As described in the introduction, in the initial phase of ECC certain domain parameters need to be specified, stored or exchanged. SEC 1 details how this should be done for systems working over the fields $\mathbb{F}_p$ and $\mathbb{F}_{2^n}$:

SEC 1: Elliptic Curve Domain Parameters over $\mathbb{F}_p$

Definition 5.2.2. Elliptic curve domain parameters over $\mathbb{F}_p$ are the sextuple:

$$\mathcal{V} := \big(p, a, b, G, \ell, c\big) \tag{5.2.1}$$

consisting of a prime p specifying the finite field $\mathbb{F}_p$, two elements $a, b \in \mathbb{F}_p$ specifying the elliptic curve $E/\mathbb{F}_p$ defined by its Weierstraß equation

$$y^2 \equiv x^3 + ax + b,$$

a base point G on $E(\mathbb{F}_p)$, a prime ℓ which is the order of $\langle G \rangle$ and an integer c which is the cofactor of the curve, $c = \#E(\mathbb{F}_p)/\ell$.

SEC 1: Elliptic Curve Domain Parameters over $\mathbb{F}_{2^n}$

Definition 5.2.3. Elliptic curve domain parameters over $\mathbb{F}_{2^n}$ are the septuple:

$$\mathcal{V} := \big(n, f(X), a, b, G, \ell, c\big) \tag{5.2.2}$$

consisting of an integer n and an irreducible binary polynomial $f(X)$ of degree n specifying the (polynomial) representation of $\mathbb{F}_{2^n}$, two elements $a, b \in \mathbb{F}_{2^n}$ specifying the elliptic curve $E/\mathbb{F}_{2^n}$ defined by its Weierstraß equation

$$y^2 + xy \equiv x^3 + ax^2 + b,$$

a base point G on $E(\mathbb{F}_{2^n})$, a prime ℓ which is the order of $\langle G \rangle$ and an integer c which is the cofactor of the curve, $c = \#E(\mathbb{F}_{2^n})/\ell$.

5.2.2 Naïve Representational Bit–Size for $\mathcal{V}$ in SEC 1

We are now in a position to calculate the required bandwidth to transmit the domain parameters given in Definitions 5.2.2 & 5.2.3 for SEC 1. This will be presented in respect to the security parameter k defined in §4.4 and the methods required by the SEC 1 for representing elements as octet strings in Definitions A.1.2, A.1.3 & A.1.4.

Lemma 5.2.4. *The worst–case naïve bit-size $\|\mathcal{V}\|$ required to represent domain parameters for prime fields $\mathbb{F}_p$ from SEC 1 for a fixed security parameter k is*

$$32\left\lceil((17/16)k+1)/8\right\rceil + 8\left\lceil(k+1)/8\right\rceil + 8\left\lceil k/128\right\rceil + 8$$

bits with point–compression and

$$40\left\lceil((17/16)k+1)/8\right\rceil + 8\left\lceil(k+1)/8\right\rceil + 8\left\lceil k/128\right\rceil$$

bits without point–compression.

Proof. We wish to compute the expected $\|\mathcal{V}\|$ for a specific security parameter $k = \lfloor \lg \ell \rfloor$.

Assume that there exists a publically known standard for the transmission of the domain parameters within $\mathcal{V}$. Then by Definition 5.2.2:

$$\|\mathcal{V}\| = \|p\| + \|a\| + \|b\| + \|G\| + \|\ell\| + \|c\|\,.$$

Since a and b are just elements of $\mathbb{F}_p$, one trivially has $\|p\| + \|a\| + \|b\| = 3\,\|p\|$. Similarly from Definition A.1.4, the generating point G can be uniquely represented using $8\left\lceil(\|p\|+1)/8\right\rceil$ or $16\left\lceil\|p\|/8\right\rceil$ bits respectively depending on whether or not point–compression is being utilised.

The cofactor c is bounded by SEC 1's recommendations for parameter generation in §3.1.1.1 of [15]. Namely that

$$c \leq 2^{s/8},$$

where 2^s is the expected number of steps required to take logarithms. Hence here $k = 2s$ and we have $c \leq 2^{k/16}$. From Definition A.1.2 one has that $\|c\| = 8\lceil k/128 \rceil$ bits.

The last parameter is ℓ. Since $(\lg \ell) \geq k$ one can safely assume that the integer ℓ requires at most a $(k+1)$–bit string to represent giving us

$$2^k \leq \ell < 2^{k+1}$$

for otherwise the system would be over specified. Hence $\|\ell\| = 8\lceil (k+1)/8 \rceil$ bits to represent as an octet string in $\mathcal{V}$.

Now we must relate the value of k to the size of p. The Haße–Weil Theorem (Theorem 4.1.8) relates the number of rational points to the field size, that is:

$$\#E(\mathbb{F}_p) = p + 1 - t \qquad \text{where} \qquad |t| \leq 2\sqrt{p}.$$

We want to ensure that the system has *at least* the security parameter k; that is $\ell \geq 2^k$. As one usually picks k first and then finds a suitable p, we follow the same approach: Fixing k and searching for the largest possible p gives us the case where the cofactor c is at a maximum and $\#E(\mathbb{F}_p)$ is minimal giving us

$$\ell \cdot c = \#E(\mathbb{F}_p) \approx \lceil p + 1 - 2\sqrt{p} \rceil .$$

Since $2^{k+1} \cdot 2^{k/16} > \ell \cdot c \geq 2^k \cdot 1$, by taking logarithms on both sides one has

$$k \leq \lg p < (17/16)k + 1.$$

Hence one needs to assume that $\max\{\lg p\} = (17/16)k + 1$ to ensure one always has a security parameter of k.

Hence one has $\|\mathcal{V}\| = 3\|p\| + 8\lceil (\|p\| + 1)/8 \rceil + (8\lceil (k+1)/8 \rceil) + 8\lceil k/128 \rceil$ bits for the compressed case. By Definition A.1.3 one has

$$\|p\| = 8\lceil (\lg p)/8 \rceil \stackrel{\max}{=} 8\lceil ((17/16)k + 1)/8 \rceil$$

and so for the (worst–case) when using point–compression this gives us

$$\begin{aligned} \|\mathcal{V}\| = 24\lceil ((17/16)k + 1)/8 \rceil + 8\lceil (8\lceil ((17/16)k + 1)/8 \rceil + 1)/8 \rceil \\ + 8\lceil (k+1)/8 \rceil + 8\lceil k/128 \rceil \text{ bits.} \end{aligned}$$

From the relation $8\lceil (8\lceil ((17/16)k + 1)/8 \rceil + 1)/8 \rceil = 8\lceil ((17/16)k + 1)/8 \rceil + 8$, the result now follows. The uncompressed case is similarly computed. □

***Remark* 5.2.5.** In Lemma 5.2.4, we truly consider the pessimistic case that one could have $c \approx 2^{k/16}$. In deployment it would be much more usual to have a small c, which would make $\ell \approx p$. This Lemma however gives the accurate worst–case assessment under SEC 1.

***Example* 5.2.6.** To place these values into context, k is usually chosen from the interval $[160, 384]$ and for these cases with prime fields $\mathbb{F}_p$ one has:

Bit Size of $\|\mathcal{V}\|$		
k	*With point compression*	*Without point compression*
160	896	1064
164	896	1064
165	928	1104
192	1056	1256
224	1216	1448
256	1408	1680
384	2088	2496

Table 5.1: SEC 1: Example Bit Sizes for $\mathcal{V}$.

For the $\mathbb{F}_{2^n}$ values one simply adds 8–bits for the compressed case and 16–bits for the other:

Lemma 5.2.7. *The worst–case naïve bit-size $\|\mathcal{V}\|$ required to represent domain parameters for binary fields $\mathbb{F}_{2^n}$ from SEC 1 for a given security parameter k is*

$$\|\mathcal{V}\| = 24\lceil((17/16)k+1)/8\rceil + 8\lceil((17/16)k+2)/8\rceil + 8\lceil(k+1)/8\rceil + 8\lceil k/128\rceil + 16$$

bits with point–compression and

$$\|\mathcal{V}\| = 40\lceil((17/16)k+1)/8\rceil + 8\lceil(k+1)/8\rceil + 8\lceil k/128\rceil + 16$$

bits without point–compression.

Proof. As before, assume that there exists a publically known standard for the transmission of the domain parameters within $\mathcal{V}$. Then by Definition 5.2.3:

$$\|\mathcal{V}\| = \|n\| + \|f(X)\| + \|a\| + \|b\| + \|G\| + \|\ell\| + \|c\|\,.$$

n is simply the degree of the polynomial $f(X)$ and is an integer in the range specified by SEC 1 in §3.1.2.1 of [15]:

$$n \in \{163, 233, 239, 283, 409, 571\}.$$

Hence $\lceil \lg 571\rceil = 10$ and one can assume that the integer n will be encoded into a double octet–string giving $\|n\| = 16$.

We shall assume the naïve (worst) case for transmitting the binary polynomial $f(X)$, whereby the n binary coefficients of f are stored in a leftmost padded octet string. Hence, $\|f(X)\| = 8\lceil n/8\rceil$ bits. Similar to the $\mathbb{F}_p$ case, one has that $\|a\| + \|b\| = 2(8\lceil n/8\rceil)$ and $\|G\| = 8\lceil(n+1)/8\rceil$ with point–compression and $16\lceil n/8\rceil$ bits without. The cofactor c here is similarly bounded by §3.1.2.1 of [15] to be $c \leq 2^{k/16}$. Hence from Definition A.1.2, $\|c\| = 8\lceil k/128\rceil$ bits.

We must now determine the last parameter ℓ. Since $(\lg \ell) \geq k$ one may safely assume that the integer ℓ requires at most a $(k+1)$–bit string to represent it or the system would be over specified. Thus $\|\ell\| = 8\lceil(k+1)/8\rceil$ bits represented as an octet–string in $\mathcal{V}$. As before we relate the value of k to the size of n using the Haße–Weil Theorem:

$$\#E(\mathbb{F}_{2^n}) = 2^n + 1 - t \qquad \text{where} \qquad |t| \leq 2^{\frac{n}{2}+1}.$$

We again want to ensure that our system has *at least* the security parameter k: Fixing k and searching for the largest possible n gives us the case where the cofactor c is at a maximum and $\#E(\mathbb{F}_{2^n})$ is at a minimum giving us

$$\ell \cdot c = \#E(\mathbb{F}_{2^n}) \approx \left\lceil 2^n + 1 - 2^{\frac{n}{2}+1} \right\rceil.$$

Since $2^{k+1} \cdot 2^{k/16} > \ell \cdot c \geq 2^k \cdot 1$, by taking logarithms on both sides one has $k \leq n < (17/16)k + 1$. Thus one assumes that $\max\{n\} = (17/16)k + 1$. Under this one can then show when using point–compression one has

$$\|\mathcal{V}\| = 24\lceil((17/16)k+1)/8\rceil + 8\lceil((17/16)k+2)/8\rceil + 8\lceil(k+1)/8\rceil + 8\lceil k/128\rceil + 16.$$

The uncompressed case is analogously computed and the result now follows. □

To give an estimate for the size required to represent the maximal $\mathcal{V}$ from SEC 1 for a given k, it is trivial to observe that:

Corollary 5.2.8. *As k grows for both prime fields $\mathbb{F}_p$ and binary fields $\mathbb{F}_{2^n}$, one requires*

$$\|\mathcal{V}\| \to \left(5 + \frac{5}{16}\right)k$$

bits for the compressed case and

$$\|\mathcal{V}\| \to \left(6 + \frac{3}{8}\right)k$$

bits for the uncompressed case.

5.3 Previous Research: Smart

As noted in the Motivation this subject has received little attention. Efficient domain parameter representation for a restricted model was considered by Smart in [76]. We investigate this further in §6.2. However, Smart made the crucial observation that to establish the order of the largest prime subgroup ℓ, one only need know the trace of the Frobenius of $E(\mathbb{F}_q)$. We place this idea under SEC 1:

Theorem 5.3.1. *Let k be a fixed security parameter and the cofactor be bounded by $c \leq 2^{k/16}$. Then using the triple $\{q, t, c\}$ one can establish the order $\ell = (q + 1 - t)/c$ in only*

$$\|t\| = \lceil((17/16)k + 1)/2\rceil + 2$$

bits with the trace of the Frobenius of $E(\mathbb{F}_q)$ instead of directly by ℓ.

Proof. Recall the Haße–Weil theorem (Theorem 4.1.8) which relates the number of rational points to the field size:

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

It is clear that the (signed) trace t and q completely determine the order $\#E(\mathbb{F}_q)$. Thus specifying the triple $\{q, t, c\}$ determines the group order $\ell = (q + 1 - t)/c$.

In the proofs of the Lemmata 5.2.4 and 5.2.7, we showed that for both prime and binary fields, $\max\{\lg q\} = (17/16)k + 1$ bits when the cofactor is allowed to be bound by $c \leq 2^{k/16}$, as when using SEC 1. Hence one can establish ℓ by sending

$$\|t\| = \lceil((17/16)k + 1)/2\rceil + 2$$

bits including the sign–bit for t by using $|t| \leq 2\sqrt{q}$. □

This is an important observation since for a given security parameter k, one expects ℓ to require $k+1$ bits to represent whereas at most one actually requires $\lceil((17/16)k+1)/2\rceil+2$ with negligible extra computation.

5.4 Reducing Redundancy in the Definition of $\mathcal{V}$

Here we present more efficient methods for establishing domain parameters than was given in Definitions 5.2.2 & 5.2.3. Specifically we show how one can establish the order ℓ and the cofactor c using analogous 'modified' versions in Sections 5.4.1 & 5.4.2 receptively. These modified versions can be used to define any order and cofactor with no loss in flexibility for fewer bits.

5.4.1 Establishing the Order ℓ for General Finite Fields

In this section we present various methods to reduce the size required to specify the domain parameter $\ell \in \mathcal{V}$.

Here we extend Smart's result from Theorem 5.3.1 and define the *modified trace*, denoted $\hat{t}$. With this we prove that at least an extra bit can always be saved for non–binary fields $\mathbb{F}_q$ and 1 or 2 fewer bits for the binary case:

Definition 5.4.1. Let the field be a power of an odd prime $\mathbb{F}_q = \mathbb{F}_{p^d}$ for $d \geq 1$, t be the trace of the Frobenius and c be the cofactor from $\mathcal{V}$. Then the $\mathbb{F}_q$*–modified trace* $\hat{t}$ is defined to be

$$\hat{t} = \begin{cases} t/2 & \text{when } c \in 2\mathbb{Z} \\ (t-1)/2 & \text{when } c \in 2\mathbb{Z}+1. \end{cases} \tag{5.4.1}$$

Lemma 5.4.2. *Let k, c, t and $\mathbb{F}_q$ be defined as above. Then using the triple $\{q, \hat{t}, c\}$ one can specify the order $\ell = (q+1-t)/c$ in only*

$$\left\|\hat{t}\right\| = \lceil((17/16)k+1)/2\rceil + 1$$

bits using the $\mathbb{F}_q$–modified trace instead of directly by ℓ.

Proof. Again by Haße–Weil one has

$$c \cdot \ell = \#E(\mathbb{F}_q) = q + 1 - t \tag{5.4.2}$$

where $|t| \leq 2\sqrt{q}$. Since p is odd, $q = p^d$ is odd for all $d \geq 1$ and similarly ℓ is also. From equation (5.4.2), one can see that t and c always share the same parity. Thus when $c \in 2\mathbb{Z}$ set $t = 2\hat{t}$, otherwise set $t = 2\hat{t}+1$.

Clearly, the modified trace $\hat{t}$ is always one bit smaller than the trace of the Frobenius of $E(\mathbb{F}_q)$. The result now follows. □

One now defines the analogous case for binary fields: It is well known that the Weierstraß form

$$E/\mathbb{F}_{2^n} : y^2 + xy = x^3 + ax + b \tag{5.4.3}$$

with $b \neq 0$ represents every isomorphism class of ordinary elliptic curves over $\mathbb{F}_{2^n}$ ([9], p37). With this, one recalls Lemma III.4 from ([9], p38) given here:

Lemma 5.4.3. *Consider an elliptic curve defined by equation* (5.4.3) *over* $\mathbb{F}_{2^n}$. *Then,*

$$\#E(\mathbb{F}_{2^n}) \equiv \begin{cases} 0 \pmod 4 & \text{if } \mathrm{Tr}_{2^n|2}(a) = 0, \\ 2 \pmod 4 & \text{if } \mathrm{Tr}_{2^n|2}(a) = 1. \end{cases} \tag{5.4.4}$$

One can now present the analogous results for binary fields:

Definition 5.4.4. Let the field be $\mathbb{F}_{2^n}$ where $n \geq 1$, t be the trace of the Frobenius and c be the cofactor from $\mathcal{V}$. Then the *modified trace* $\hat{t}$ is defined by:

$$\hat{t} = \begin{cases} (t-1)/2 & \text{when } \mathrm{Tr}_{2^n|2}(a) = 1, \\ (t-1)/4 & \text{when } \mathrm{Tr}_{2^n|2}(a) = 0. \end{cases} \tag{5.4.5}$$

Lemma 5.4.5. *Let k be a fixed security parameter, the cofactor be bounded by $c \leq 2^{k/16}$ and the field be $\mathbb{F}_{2^n}$. Then using the quadruple $\{n, a, \hat{t}, c\}$ one can specify the order $\ell = (2^n + 1 - t)/c$ of the elliptic curve in equation* (5.4.3) *in*

$$\left\|\hat{t}\right\| = \lceil ((17/16)k + 1)/2 \rceil + \mathrm{Tr}_{2^n|2}(a)$$

bits using the modified trace instead of directly by ℓ.

Proof. Again the Haße–Weil theorem gives us

$$c \cdot \ell = \#E(\mathbb{F}_q) = 2^n + 1 - t \tag{5.4.6}$$

where $|t| \leq 2^{(n/2)+1}$. From Lemma 5.4.3 we have

$$\#E(\mathbb{F}_q) \equiv 2 \cdot \mathrm{Tr}_{2^n|2}(a) \pmod 4.$$

Since ℓ is a large prime, $\ell \not\equiv 0, 2 \pmod 4$ and so $c \geq 2$ where $c \equiv 2^{2-\mathrm{Tr}_{2^n|2}(a)} \pmod 4$ itself. Thus since the LHS of equation (5.4.6) contains a factor of $2^{2-\mathrm{Tr}_{2^n|2}(a)}$ so must the RHS, which gives us that $(t-1) \equiv 2^{2-\mathrm{Tr}_{2^n|2}(a)} \pmod 4$.

Hence, define the signed integer $\hat{t}$ to be the modified trace given by

$$\hat{t} = (t-1) \cdot 2^{\mathrm{Tr}_{2^n|2}(a)-2}.$$

This is what was given in Definition 5.4.4. Hence by using $\hat{t}$ in lieu of t from Theorem 5.3.1, the result now follows. □

5.4.2 The Modified Cofactor

When the modified trace $\hat{t}$ does not directly depend on the cofactor c to recover the original trace t, one can save extra bits in the representation of $\mathcal{V}$. We do this by defining the *modified cofactor*, $\hat{c}$. This is marginally beneficial for general binary fields here but especially useful when using subfield curves later in §5.6.4.

Definition 5.4.6. Let the field be $\mathbb{F}_{2^n}$ for $n \geq 1$, $\hat{t}$ the modified trace given in Definition 5.4.4 and a and c as defined in $\mathcal{V}$. Then the *modified cofactor* $\hat{c}$ is

$$\hat{c} := \begin{cases} c/2 & \text{when } \mathrm{Tr}_{2^n|2}(a) = 1, \\ c/4 & \text{when } \mathrm{Tr}_{2^n|2}(a) = 0. \end{cases} \tag{5.4.7}$$

Lemma 5.4.7. *The expected bit size to establish the cofactor $c \in \mathcal{V}$ is $\|c\| = 8\lceil k/128 \rceil$ bits in SEC 1 (Lemmata 5.2.4). Using the modified cofactor which requires negligible additional computation, one requires:*

$$\|\hat{c}\| = \begin{cases} 8\lceil (k-16)/128 \rceil & \text{when } \mathrm{Tr}_{2^n|2}(a) = 1, \\ 8\lceil (k-32)/128 \rceil & \text{when } \mathrm{Tr}_{2^n|2}(a) = 0 \end{cases} \tag{5.4.8}$$

bits over $\mathbb{F}_{2^n}$ to establish c.

Proof. By direct computation from Definition 5.4.6. □

5.5 Excluding Redundancy from $\mathcal{V}$ for SEC 1

Here we present the true cost of SEC 1 if it were to use the principles of modified traces and cofactors. As this does not restrict SEC 1, this may be considered the 'best case' we know of, and expect, using the standard outlined in [15]. It also serves to highlight the redundancy present in this construction.

Corollary 5.5.1. *The worst–case bit–size $\|\mathcal{V}\|$ required to represent domain parameters for prime fields $\mathbb{F}_p$ from SEC 1 when using the modified trace, $\hat{t}$, for a given security parameter k is*

$$32\lceil ((17/16)k+1)/8 \rceil + 8\lceil (\lceil ((17/16)k+1)/2 \rceil + 1)/8 \rceil + 8\lceil k/128 \rceil$$

bits with point–compression and

$$40\lceil ((17/16)k+1)/8 \rceil + 8\lceil (\lceil ((17/16)k+1)/2 \rceil + 1)/8 \rceil + 8\lceil k/128 \rceil$$

bits without.

Proof. One merely notes that we do not transmit ℓ at a cost of $8\lceil (k+1)/8 \rceil$ bits, but instead use $\hat{t}$ which requires $\lceil ((17/16)k+1)/2 \rceil + 1$ bits to represent in all cases. This integer is subsequently padded as an octet–string as defined by SEC 1. □

Corollary 5.5.2. *The worst–case bit–size $\|\mathcal{V}\|$ required to represent domain parameters for binary fields $\mathbb{F}_{2^n}$ from SEC 1 when using the modified trace and cofactor, for a given security parameter k is*

$$\begin{aligned} \|\mathcal{V}\| = 32\lceil ((17/16)k+1)/8 \rceil + 8\lceil (\lceil ((17/16)k+1)/2 \rceil + 1)/8 \rceil \\ + 8\left\lceil (k - 16(2 - \mathrm{Tr}_{2^n|2}(a)))/128 \right\rceil \end{aligned}$$

bits with point-compression and

$$\begin{aligned} \|\mathcal{V}\| = 40\lceil ((17/16)k+1)/8 \rceil + 8\lceil (\lceil ((17/16)k+1)/2 \rceil + 1)/8 \rceil \\ + 8\left\lceil (k - 16(2 - \mathrm{Tr}_{2^n|2}(a)))/128 \right\rceil \end{aligned}$$

bits without.

Proof. Analogous to the proof of Corollary 5.5.1 but also by adding the binary result for the modified cofactor given in Lemma 5.4.7. □

To give an estimate of the size required to represent $\mathcal{V}$ from SEC 1 with the indicated modifications for a given k, it is trivial to observe that:

Corollary 5.5.3. *As $k \to \infty$ for both prime and binary fields $\mathbb{F}_q$ we require*

$$\|\mathcal{V}\| \to \Big(4 + \frac{13}{16}\Big)k$$

bits when using point–compression and

$$\|\mathcal{V}\| \to \Big(5 + \frac{7}{8}\Big)k$$

bits when not.

Corollary 5.5.4. *For both prime and binary fields $\mathbb{F}_q$ under SEC 1 given in* [15]*, one can show that the redundancy in the definition of $\mathcal{V}$ approaches*

$$\left(\frac{k}{2}\right) \text{ bits as } k \to \infty$$

with or without point–compression.

5.6 Subfield Curves under SEC 1

In this section we consider a special type of elliptic curve which is often used in cryptography; the *subfield curve*. A well known example of these are Koblitz curves defined in §4.2. Koblitz curves are commonly implemented and subfield curves over non–binary extension fields are increasingly used with smart–cards. For these cases we present how modified traces and cofactors are particularly efficient to represent. This will form a foundation for the research undertaken in Chapter 6.

5.6.1 Establishing the Order ℓ when using Subfield Curves

Definition 5.6.1. Let $\mathbb{F}_q$ be a finite field of any characteristic. Let E be an elliptic curve defined over $\mathbb{F}_q$ be lifted over an extension field $\mathbb{F}_{q^m}$ for $m > 1$. Then $E/\mathbb{F}_{q^m}$ is denoted a *subfield curve*.

We now explain Weil's method to compute the trace of the Frobenius of $E(\mathbb{F}_{q^m})$ from the trace of $E(\mathbb{F}_q)$, ([75], p132). This can be computed efficiently using only integer arithmetic and it will allow us to define the group order ℓ far more compactly for these cases.

Theorem 5.6.2. *Let the curve $E(\mathbb{F}_q)$ have order $\#E(\mathbb{F}_q) = q + 1 - t$ where $t = \alpha + \beta$ is the trace of the q^{th}–power Frobenius and*

$$X^2 - tX + q = (X - \alpha)(X - \beta) \tag{5.6.1}$$

is its characteristic polynomial with $\alpha = \beta^* \in \mathbb{C}$.

Then the trace of the q^m–th power Frobenius is given by $t_m = \alpha^m + \beta^m$ and the order of the lifted curve $E(\mathbb{F}_{q^m})$ is

$$\#E(\mathbb{F}_{q^m}) = q^m + 1 - t_m.$$

Proof. See ([75], p135). □

An important observation is that $\alpha^m + \beta^m$ is an integer which can be computed using solely arithmetic in $\mathbb{Z}$:

Lemma 5.6.3. *Let $t_n = \alpha^n + \beta^n$ with $t_0 = 2$, $t_1 = t$. Then*

$$t_{n+1} = t_1 t_n - q t_{n-1}$$

for all $n \geq 1$.

Proof. Equation (5.6.1) gives us $\alpha^2 - t\alpha + q = 0$ which upon multiplication by α^{n-1} gives

$$\alpha^{n+1} = t\alpha^n - q\alpha^{n-1}.$$

Computing the analogous result for β and summing gives

$$\alpha^{n+1} + \beta^{n+1} = t(\alpha^n + \beta^n) - q(\alpha^{n-1} + \beta^{n-1})$$

and we are done. For a more general proof, see Corollary 8.3.3 later. □

As this calculation is fast, all one needs to send is the trace of the curve $E(\mathbb{F}_q)$ to establish the order $\#E(\mathbb{F}_{q^m})$. This requires approximately $\frac{k}{2m}$ bits as opposed to $k + 1$. We now present the details:

Over Odd Characteristic Fields

Definition 5.6.4. Let q be a power of an odd prime p. Let $E/\mathbb{F}_{q^m}$ be a subfield curve where $E/\mathbb{F}_q$, $m > 1$ and $\hat{t}_1$ the modified trace of the Frobenius of $E(\mathbb{F}_q)$, given in Definition 5.4.1. Then the *modified trace* $\hat{t}$ of $E/\mathbb{F}_{q^m}$ is defined to be

$$\hat{t} := \hat{t}_1. \tag{5.6.2}$$

Lemma 5.6.5. *Let $\mathbb{F}_{q^m}$ be as defined above and k a fixed security parameter. Then using the quadruple $\{q, m, \hat{t}, c\}$ one can establish the order $\ell = \#E(\mathbb{F}_{q^m})/c$ in only*

$$\left\|\hat{t}\right\| = \lceil((17/16)k + 1)/2m\rceil + 1$$

bits using the modified trace instead of directly representing by ℓ.

Proof. It is clear from Theorem 5.6.2 and Lemma 5.6.3 that using the triple $\{q, m, t_1\}$ one can compute the order of $E(\mathbb{F}_{q^m})$. Since $c \in \mathcal{V}$ one can establish ℓ with only computation in $\mathbb{Z}$. Thus the method is well–defined.

We now need to establish $\left\|\hat{t}\right\|$. From Lemma 5.4.2 and $|\hat{t}_1| \leq 2\sqrt{q}$ one has that

$$\left\|\hat{t}_1\right\| = \lceil((17/16)k + 1)/2m\rceil + 1.$$

□

Over Binary Fields

As before we send a modified trace which is the trace of the subfield curve $E/\mathbb{F}_{2^d}$. However due to characteristics of curves over binary fields, one can do even better when establishing ℓ leading to fewer bits being required.

Definition 5.6.6. Define the field $\mathbb{F}_{2^d}$ for $d \geq 2$ and its extension $\mathbb{F}_{2^{dm}}$ for an $m > 1$. Let $E/\mathbb{F}_{2^d}$ and $\hat{t}_1$ be its modified trace given in Definition 5.4.4. Then the *modified trace* $\hat{t}$ of $E/\mathbb{F}_{2^{dm}}$ is defined to be

$$\hat{t} := \hat{t}_1.$$

Lemma 5.6.7. *Let k be a fixed security parameter and $E/\mathbb{F}_{2^d}$ and $\mathbb{F}_{2^{dm}}$ be defined as above. Then using the quadruple $\{q, m, \hat{t}, a\}$ one can establish the order $\ell = \#E(\mathbb{F}_{2^{dm}})/c$ in only*

$$\left\|\hat{t}\right\| = \lceil((17/16)k + 1)/2m\rceil + \mathrm{Tr}_{2^d|2}(a) \text{ bits.}$$

When $d = 1$, $E/\mathbb{F}_2$ is a Koblitz curve and here one does not require sending a trace or modified trace at all to establish $\#E(\mathbb{F}_{2^m})$. Hence here

$$\|t\| = \left\|\hat{t}\right\| = 0.$$

Proof. This proof is analogous to the one of Lemma 5.6.5, but with the realisation that the subfield curve $E/\mathbb{F}_{2^d}$ must also must have a factor of 2 or 4 in its group order by Lemma 5.4.3. Hence we use the modified trace $\hat{t}_1 = (t_1 - 1) \cdot 2^{\mathrm{Tr}_{2^d|2}(a)-2}$ of the curve over $\mathbb{F}_{2^d}$ rather than the modified version of the full trace. With this one can use the recurrence formula to compute the full trace and subsequently the order of the lifted curve as $\#E(\mathbb{F}_{2^{dm}}) = 2^{dm} + 1 - t_{dm}$ where

$$t_{dm+1} = (\hat{t}_1 \cdot 2^{2-\mathrm{Tr}_{2^d|2}(a)} + 1)t_{dm} - qt_{dm-1}.$$

In the case of Koblitz curves things are simpler: When one defines subfield curves E over $\mathbb{F}_2$ and then lifts them to $\mathbb{F}_{2^m}$, as $b \neq 0$ in the Weierstraß equation there exists only two possible curves corresponding to $a \in \{0, 1\} \simeq \mathbb{F}_2$. When $a = 0$, $\#E_0(\mathbb{F}_2) = 4$ and hence the trace of E_0 is -1. Similarly; $E_1(\mathbb{F}_2) = 2$ implies $t_1(E_1) = 1$. Thus $t_1 = 2a - 1$ and so nothing needs to be sent as the trace is implicit from the Weierstraß form. The lifted trace can now be computed using the recurrence relation detailed above. □

5.6.2 Performance Considerations

***Remark* 5.6.8.** The cofactor c by definition should be small, namely; $c \leq 2^{k/16}$. Hence the computation required to compute the order $\ell = (q + 1 - t)/c$ is generally inexpensive. When $E/\mathbb{F}_{2^n}$ and c is a power of two, then the computation required to compute the order $\ell = (2^n + 1 - t)/c$ is negligible.

***Remark* 5.6.9.** When working with the subfield modified trace, there is an additional cost using Lemma 5.6.3 to compute the lifted group order.

However, this recurrence relation requires n steps each consisting of two multiplications and two additions in $\mathbb{Z}$. For a field $\mathbb{F}_{q^m}$ of any characteristic, set $n = \lceil \lg(q^m) \rceil$. Since one has $\max\{|t|\} \leq 2\sqrt{q^m} \Rightarrow \|(|t|)\| \leq (n/2) + 1$ bits. Thus the recurrence relation requires

$$h\big(n \cdot (2(n/2 + 1)^2 + 2(n/2 + 1))\big) = O(n^3)$$

steps for some $h \in \mathbb{R}_{>0}$, which is a polynomial–time algorithm.

5.6.3 Modified Trace Unsigning when Using Verification

The sign bit of $\hat{t}$ in all the above cases (Sections 5.4.1 & 5.6.1) may be dropped for a negligible additional computational cost when a protocol insists on *point–verification*: When a device receives the domain parameters $\mathcal{V}$, it verifies that the ephemeral base point $G \in \mathcal{V}$ is in fact of the correct order ℓ, to avoid small subgroup attacks like those detailed in ([16], p569).

When using point–verification, SEC 1 recommends testing that ℓ is a prime (with some high confidence) and $[\ell]G = \mathcal{O}$. This leads to the following results:

Definition 5.6.10. Let $\mathcal{P}_r(x)$ be the Miller–Rabin primality test for a given odd integer $x \geq 3$ with r trials, [63]. $\mathcal{P}_r(x)$ returns `false` when x is composite with probability of 1. $\mathcal{P}_r(x)$ returns `true` when x is probably prime with confidence of failure of 4^{-r}.

Definition 5.6.11. Let $\mathbb{F}_q$ be a finite field, $E/\mathbb{F}_q$ an ordinary elliptic curve,

$$t = q + 1 - \#E(\mathbb{F}_q) = q + 1 - c\ell \neq 0$$

with cofactor c and ℓ the largest prime dividing $\#E(\mathbb{F}_q)$. One then recovers the original order ℓ using $|t|$ by evaluating the following:

`A:` **set** $\alpha \leftarrow q + 1 - |t|$ and $\beta \leftarrow q + 1 + |t|$.

`B:` **if** $c \neq 1, 2$ **then**:

- **if** $\big(\alpha \pmod c \neq 0$ and $\beta \pmod c = 0\big)$ **set** $\ell \leftarrow \beta/c$.
- **else if** $\big(\alpha \pmod c = 0$ and $\beta \pmod c \neq 0\big)$ **set** $\ell \leftarrow \alpha/c$.
- **else if** $\big(\alpha \pmod c = 0$ and $\beta \pmod c = 0\big)$ **goto step** `C`.
- **else reject** $\mathcal{V}$**. halt**.
- **if** $\mathcal{P}_r(\ell) =$ `false` **reject** $\mathcal{V}$**. halt**.
- **goto step** `G`.

`C:` **else** $\alpha \equiv \beta \pmod c$. **set** $\ell \leftarrow \alpha/c$.

`D:` **if** $\mathcal{P}_r(\ell) =$ `true` **goto step** `G`.

`E:` **set** $\ell \leftarrow \beta/c$. **if** $\mathcal{P}_r(\ell) =$ `true` **goto step** `G`.

`F:` **reject** $\mathcal{V}$**. halt**.

`G:` **return** `probably prime`.

As a final verification one can check whether or not $[\ell]G = \mathcal{O}$. This however could be computationally expensive, especially when $c \neq 1$ for odd characteristic fields and when $c \pmod 2 \not\equiv 0$ for binary ones.

Lemma 5.6.12. *The constructive method provided in Definition 5.6.11 allows one to construct the order ℓ using the triple $\{q, |t|, c\}$ to any confidence level desired. Additionally it verifies that the order ℓ is in fact valid.*

Proof. Assume E is not supersingular and thus the trace is non–zero. α and β are two possibilities for the order $\#E(\mathbb{F}_q)$. If only one is divisible by c there is no ambiguity. Else, with a high probability one of α/c or β/c is prime. Here we select the smallest possible candidate $\ell = \alpha/c$. If $\mathcal{P}_r(\ell) =$ `false` one knows α/c is composite and sets $\ell = \beta/c$ and retests $\mathcal{P}_r(\ell)$. If `false` again, we reject $\mathcal{V}$ since neither α nor β is prime and we do not have a valid order.

Otherwise, we have arrived at `step G` and we have that our candidate ℓ is composite with probability $1/4^r$, hence we assume ℓ is prime. □

***Remark* 5.6.13.** An expensive but often recommended final test would be to test if $[\ell]G = \mathcal{O}$ for our computed ℓ. Clearly both $\ell_\alpha = \alpha/c$ and $\ell_\beta = \beta/c$ cannot simultaneously be prime divisors of $\#E(\mathbb{F}_q)$ since $\ell_\alpha \ell_\beta \gg \#E(\mathbb{F}_q)$ because $c \leq 2^{k/16}$. Hence with the same probability that ℓ is not a pseudoprime, one has that ℓ is the correct prime order of G.

A natural question that arrises is: *Do the methods presented for the signed modified trace work for the unsigned variant?* We now answer this question:

Lemma 5.6.14. *For the Lucas sequence given in Lemma 5.6.3 one has*

$$t_{n+m} = t_n t_m - q^m t_{n-m} \tag{5.6.3}$$

for integers $n > m > 0$.

Proof. For any two numbers α and β and integers $n > m > 0$ one has

$$(\alpha^n + \beta^n)(\alpha^m + \beta^m) = (\alpha^{n+m} + \beta^{n+m}) + \alpha^m \beta^m (\alpha^{n-m} + \beta^{n-m}).$$

Hence for the Lucas sequence above:

$$t_{n+m} = (\alpha^{n+m} + \beta^{n+m}) = t_n t_m - (\alpha\beta)^m t_{n-m} = t_n t_m - q^m t_{n-m}.$$

□

Theorem 5.6.15. *Let $|\alpha^n + \beta^n| = |t_n|$, $t_0 = 2$ and $t_1 = |t|$. Then*

$$t_{n+1} = t_1 t_n - q t_{n-1} \tag{5.6.4}$$

for all $n \geq 1$.

Proof. We want to show that the Lucas sequence given in equation (5.6.4) is equivalent up to sign to the one from Lemma 5.6.3, which we shall denote t'_n. Clearly this holds when $t > 0$ since $t = |t|$. Hence we consider $t < 0$ so that $t = -|t|$ here. Evaluating the two relations for the first three iterations gives:

n	t_n	t'_n
0	2	$2 = t_0$
1	t	$\|t\| = -t = -t_1$
2	$t^2 - 2q$	$\|t\|^2 - 2q = t_2$
3	$t(t^2 - 3q)$	$\|t\|(t^2 - 3q) = -t_3$

Using equation (5.6.3) one has that

$$t_{n+2} = t_2 t_n - q^2 t_{n-2}$$

which one can use to show that $t'_4 = t'_2 t'_2 - q^2 t'_0 = t_4$. For $n = 4, 6, \ldots$ one can show by induction that for all $n \in 2\mathbb{N}$ one has that $t'_n = t_n$. Likewise for odd n one has

$$t'_5 = t'_2 t'_3 - q^2 t'_1 = t_2(-t_3) - q^2(-t_1) = -t_5$$

which again by induction allows us to show that $t'_n = -t_n$ when n is odd. Hence $|t'_n| = |t_n|$. The result now follows. □

Corollary 5.6.16. *Given the absolute value of the modified trace $|\hat{t}|$ and the cofactor c, one is able to construct the absolute value of the full trace, $|t|$.*

***Remark* 5.6.17.** We stress that this extra bit saved here is *optional*: If verification is non–standard then you pay a computational penalty when accepting $\mathcal{V}$. For more on verification, see ([78], p18).

5.6.4 The Modified Cofactor for Subfield Curves

Analogous to §5.4.2, when the computation of the full trace from the modified trace does not depend on c, one can save extra bits in the representation of $\mathcal{V}$.

Definition 5.6.18. Let $\mathbb{F}_q$ be a finite field, $E/\mathbb{F}_q$ be a subfield curve and $c \in \mathcal{V}$ be the cofactor of the lifted curve $E(\mathbb{F}_{q^m})$. Then the *modified cofactor* $\hat{c}$ is defined by

$$\hat{c} := c/(\#E(\mathbb{F}_q)).$$

Corollary 5.6.19. *The expected bit–size to establish the cofactor $c \in \mathcal{V}$ is $\|c\| = 8\lceil k/128 \rceil$ bits in SEC 1 (Lemmata 5.2.4 & 5.2.7). Using the modified cofactor which requires negligible additional computation, one merely requires:*

$$\|\hat{c}\| = \begin{cases} 8\lceil (k - 16\lg q)/128 \rceil & \text{when } \mathrm{char}(\mathbb{F}_q) \neq 2, \\ 8\lceil (k - 16d)/128 \rceil & \text{when } q = 2^d,\ d > 2 \end{cases} \tag{5.6.5}$$

bits when using subfield curves to establish c.

Proof. From Lagrange one knows the group order of the subfield curve $N_1 = \#E(\mathbb{F}_q)$ always divides that of the lifted curve $N_m = c \cdot \ell = \#E(\mathbb{F}_{q^m})$. Since ℓ is prime, $N_1 \mid c$. Thus we only require $\hat{c} = c/N_1$ to establish c. □

5.7 Discussions & Conclusions

SEC 1 is not the only standard in common use, however it is representative of how domain parameters are often defined. In our work here, we analysed the best case information rate for SEC 1: Without changing what was specifiable by the model (the specific curves, fields et cetera) how many bits did one require to represent these using compressed versions of domain parameter elements.

In §5.3 we presented Smart's result that one could specify the group order ℓ by sending the curves trace t. This requires approximately half the bits than sending ℓ directly. In §5.4.1 & 5.4.2 we gave definitions of the modified trace and modified cofactor of the curve. These save a few bits over what has been previously published to specify the trace and cofactor respectively. We presented the worst–case bit–size of domain parameters for SEC 1 in §5.5. These results

although lower than previously published are asymptotically the same as what one gets by using Smart's result: $k/2$ bits lower for a security parameter k.

Should one wish to do better, one needs to consider an improved model for specifying domain parameters. §5.6.1 gave results on how one could establish the order of subfield curves far more efficiently than previously published. Subfield curves are indirectly representable by the SEC 1 model but not optimised for it as our results show. This motivates us to search for an improved model, especially since smart–cards and restricted devices often use such fields. It is in these constrained environments where bandwidth is most important. We consider this approach next in Chapter 6.

The lunatic, the lover, and the poet,
Are of imagination all compact.

[Act V, Scene I, A Midsummer Night's Dream]
William Shakespeare (1564–1616)

6 Compact Domain Parameters

6.1 Motivation

When using ECC one must specify domain parameters, namely

$$\mathcal{V} := (\mathbb{F}_q, E/\mathbb{F}_q, \ell, c, P)$$

as defined in the prequel. For some applications these are fixed for all users, hence are never communicated and are hardwired into the implementation. Other applications are more general, since a fixed model is restrictive and often undesirable as discussed in §5.1. This is what we consider here. We assume that these parameters must be communicated and as such the bandwidth ought to be minimal.

The fundamental question one asks oneself is: *How general should a system be?* In Chapter 5 we investigated how one can reduce domain parameter bandwidth, under a commonly used standard (SEC 1), without restricting the families constructible.

The alternative is to restrict the families definable to popular choices, where one uses certain fields and curves which allow faster group arithmetic. For example; NIST in [57] recommends fixing $a = -3$ in the Weierstraß form for curves over non–binary fields. This is because it accelerates the group arithmetic when using Jacobian projective coordinates, ([16], p282). Similarly, since the arithmetic performance of $\mathbb{F}_p$ depends on the underlying prime p, one is often recommended to use primes of certain forms (viz. [5, 14, 36, 57, 76, 80, 88]). Since we assume here that one cannot embed an ECDLP into an $\mathbb{F}_q$–DLP, improved SNFS attacks (e.g. the Frey–Rück attack [27]) are not relevant. Hence there are no known weaknesses in using these forms. Since these parameters are non–random, the key idea is that one may be able to transmit them more efficiently, reducing bandwidth.

In this chapter, we present an efficient way to represent domain parameters for certain restricted families, denoted *compact domain parameters*. These will be particularly suitable for multi–curve environments. Moreover, we consider efficient domain parameter representation at a higher level: If a multi–curve environment required t curves all with a security parameter of k, which is the most bandwidth efficient system to use?

This is an important observation which will allow us to compare the various systems one can use, in terms of the amount of curves/fields available per bits required to represent them, for a given security parameter. Thus in situations where domain parameter bandwidth is important (e.g. large multi–curve

environments), one can choose a system that is fit–for–purpose and not over–specified, giving the optimum systems–per–bit.

In our work here, we are not concerned with issues of whether some fields may prove to be weaker in the future (e.g: *Optimal Extension Fields* given in §6.5.5). We are only considering currently proposed systems (fields and curves), and attempting to represent them more compactly.

6.2 Previous Research: Smart

Compressing domain parameters has received little attention: We are only aware of the work by Smart in [76] précised here and that by Brown, Myers & Solinas [11] given in §6.3.

Efficient domain parameter representation was considered by Smart in [76], where methods were suggested for binary and prime fields of size $[150, 255]$ bits. Smart noted that for binary fields, one could use the minimal polynomial trinomial

$$X^n + X^c + 1$$

where c would require 8–bits and the odd n only 7–bits to define. For prime fields $\mathbb{F}_q$, he restricted the choice to primes of the form

$$q = 2^n + c$$

where $1 \leq c \leq n \leq 255$. Listing all such primes q he noted there were 174. Here representing q also requires 15–bits since $\|n\| = 8$ and $\|c\| = 7$ since c must be odd for q to be prime. Hence by using a single additional bit indicating the parity of the characteristic, one is able to define the field with only 16–bits.

The elliptic curve in [76] is defined in exactly $n + 8$ bits. This is done by restricting the size of the coefficients chosen (or randomly generated by computing isomorphisms) to define its Weierstraß form. Such a curve was only accepted if it had what we coin, a *minimal cofactor*:

Definition 6.2.1. The *minimal cofactor* c of an elliptic curve $E/\mathbb{F}_q$ is defined by

$$c := \begin{cases} 1 & \text{when } \mathrm{char}(\mathbb{F}_p) \neq 2, \\ 2(2-a) & \text{when } q = 2^n. \end{cases}$$

Thus in all cases, the minimal cofactor is simply the smallest cofactor possible.

The crucial observation of Smart was that to establish the order of the largest prime subgroup ℓ, one only need know the trace of the Frobenius of $E(\mathbb{F}_q)$. This result was present earlier in Theorem 5.3.1 of §5.3. Using this one requires $\approx n/2 + 2$ bits as opposed to an expected $n + 1$. Finally the generating point was represented by a random 7–bit string which when padded with $n - 7$ zero–bits, represented an abscissa of an elliptic point of order ℓ. This could be padded into a single byte–string where the extra bit represented which ordinal was being used in point decompression.

As a product of this, Smart was able to represent the domain parameters for his restricted system using only

$$\|\mathcal{V}\| = \|f\| + \|E/\mathbb{F}_q\| + \|\ell\| + \|G\| = 8\lceil (n+8)/8 \rceil + 8\lceil (n+4)/16 \rceil + 24 \text{ bits.}$$

When $n \in 8\mathbb{N}$, this form is particularly efficient requiring only 328 bits to define a 192–bit system, approximately $1.7k$ bits.

Observations

The work by Smart in [76] in part motivates the work we have undertaken here. Smart observed that one only needs send the trace of the Frobenius of $E(\mathbb{F}_q)$ rather than the whole order ℓ. This observation is clearly system independent and requires negligible or no extra computation for a useful bandwidth saving.

Smart then suggested how one could define a restricted system for a multi–curve environment: This system only allowed prime and binary fields and did not include extension and optimal extension fields (OEFs) which are becoming more popular. One could only define 174 prime fields, which may prove insufficient, and the key–sizes are contained within a small fixed range which is unsuitable for long–term usage.

6.3 Previous Research: Brown, Myers & Solinas

Brown, Myers & Solinas in [11] described a compact way of defining domain parameters by using *compact curves* $E/\mathbb{F}_p$. A Type–I compact curve is defined by:

Definition 6.3.1. Let $f \equiv 2 \pmod 3$ and $g \equiv 3 \pmod 6$ be integers such that

$$p := f^2 - fg + g^2 \qquad \text{and} \qquad r := p + 1 - (2f - g)$$

are both prime. Then define the *compact curve* E and corresponding base point P as:

$$E/\mathbb{F}_p := \begin{cases} y^2 = x^3 - 2, & P = (3, 5) \quad \text{if } p \equiv 7 \pmod 8 \\ y^2 = x^3 + 2, & P = (-1, 1) \quad \text{otherwise.} \end{cases}$$

Finally, if $p \equiv 1 \pmod 8$ we discard the computation and choose different parameters f and g.

Proposition 6.3.2. *The compact curve $E/\mathbb{F}_p$ defined above has order $r = \#E(\mathbb{F}_p)$.*

The analogous Type–II case may be found in [11], where this compact curve has order $2q$ for some prime q. Since Type–I curves have prime orders, this is the most interesting case for us here. Thus here we analyse the bandwidth required for $\mathcal{V}$ for Type–I, which is work which is missing from [11][1].

For a security parameter $k = 2\overline{k} \in 2\mathbb{N}$, fix a value $0 < h \leq 2^{\overline{k}-4}$. The Type–I elliptic curve may be constructed as follows:

Algorithm 6.3.1 (Computing type–I compact curve). `INPUT:` $\overline{k}, h \in \mathbb{N}$
`1. compute a value` $\varepsilon \in \{0, 1, 2\}$ `such that`

$$f = 3 \cdot 2^{\overline{k}-2} + 4h + \varepsilon \equiv 2 \pmod 3;$$

[1] The bandwidth required by Type–II is approximately that of Type–I.

2. evaluate the polynomial $g(s) = 3(2s+1)$ for an $s \in \{0, 1, 2, \ldots\}$ until the corresponding pair (p, r):

$$p := f^2 - fg + g^2, \qquad r := p + 1 - (2f - g)$$

   are both prime and $p \not\equiv 1 \pmod 8$;
3. if $(p \equiv 7 \pmod 8)$ {
4. $E \leftarrow y^2 = x^3 - 2$, $P \leftarrow (3, 5)$;
5. }
6. else {
7. $E \leftarrow y^2 = x^3 + 2$, $P \leftarrow (-1, 1)$;
8. }
9. OUTPUT: E, P;

In [11] this was placed in the ID–based setting by defining h to be the integer representation of the output of a $(\overline{k} - 4)$–bit cryptographic hash–function $\mathcal{H}$. However, the curve parameters should be public and tied to an ID, thus we do not require $\mathcal{H}$ to be preimage resistant, only collision resistant. This allows us to consider $h = \mathcal{H}(\mathrm{ID}_{\mathrm{dom}})$ merely as a integer that specifies the underlying field and curve. Depending on the number of curves required, h is sized accordingly.

Trivially here g is expected to be small. So in lieu of sending (p, r) one could send $(\overline{k}, h, s)$ since $g = 3(2s + 1)$ and ε is trivially computed. This gives us

$$\|f\| = \left\|\overline{k}\right\| + \|h\| + \|s\|$$

bits. We must now estimate the size of s. When using Type–I, one increases s until $p = h_1(f, g)$ and $r = h_2(f, g)$ are both prime and $p \not\equiv 1 \pmod 8$. We require the following:

Theorem 6.3.3 (Legendre's approximation)**.** *The number of primes $\pi(x)$ less than of equal to x is*

$$\pi(x) \approx \frac{x}{\ln x - B} = \frac{rx}{\lg x - rB}$$

where $B = 1.08366$ and $r = \lg e$. This is asymptotically as good as Chebyshev's approximation of $\pi(x) \approx \mathrm{li}(x)$, [65].

Corollary 6.3.4. *The probability that a random k–bit integer q is prime is*

$$\mathbb{P}_k \approx \frac{r}{k - rB}.$$

Proof. The unsigned integer q is a k–bit number if and only if $2^{k-1} \leq q \leq 2^k - 1$. From Theorem 6.3.3 we expect the number of primes N which lie in the range

$$[x, y] = [2^{k-1}, 2^k - 1]$$

is

$$N = r\frac{2^k - 1}{\lg(2^k - 1) - rB} - r\frac{2^{k-1}}{\lg 2^{k-1} - rB} \approx \frac{r2^{k-1}}{k - rB}.$$

Since there are $T = 2^{k-1} - 1$ integers in the range $[x, y]$ one expects the probability that a random integer $q \in [x, y]$ is prime is N/T and the result now follows. □

The probability that an odd integer $p \not\equiv 1 \pmod 8$ is $\frac{3}{4}$. Since we require both p and r to be prime, one expects $\mathbb{P}_k^{-2}$ trials implying that

$$s \leq \frac{4}{3} \cdot \left(\frac{k - rB}{r} \right)^2.$$

Hence one can assume that at most,

$$\|s\| = 8 \left\lceil \frac{0.4150 + 2(\lg(k - rB) - \lg r)}{8} \right\rceil \leq 8 \left\lceil \frac{2 \lg k - 0.643}{8} \right\rceil \leq 8 \lceil (\lg k)/4 \rceil$$

which is approximately twice the size of the logarithm of $\|r\|$.

Thus one has compact domain parameters here of $\mathcal{V}_{\mathrm{BMS}} := (\overline{k}, h, s)$. The expected size for these parameters is

$$\|\mathcal{V}_{\mathrm{BMS}}\| = 8 \lceil (\lg k - 1)/8 \rceil + 8 \lceil (\lg k)/4 \rceil + (k - 4) \text{ bits.}$$

For cryptographically interesting sized groups, for example $k = 192$, this would require only 212 bits. Note that the curves produced are always ordinary when $p > 3$, since the discriminant of E is $4(a^2) + 27(b^2) \equiv 108 \pmod p$.

Note that here we choose $k = 2\overline{k} \in 2\mathbb{N}$ since this is optimal case for Brown et al's work. This analysis of the parameter sizes is not discussed in [11].

Observations

The main limitations of Brown et al's approach are flexibility and the special form of the resulting system: The system presented in [11] is very favorable in terms of the size of $\mathcal{V}_{\mathrm{BMS}}$. However, although there exists no known weakness for compact curves, the fact that the field and group order are so strongly bound makes both the Type–I and Type–II systems very specialised, something which is often undesirable. Additionally there is no known analogue of Compact Curves for binary fields, which are the fields used most often in practice.

The system presented above does have the advantage that one can tailor the size of $\mathcal{V}$, depending on how many curves a system is required to define, if one was to redefine h. This is not considered in [11] and will be investigated further in the sequel. An additional advantage with both compact curves, is by construction they admit improved group operation performance through automorphisms from Gallant, Lambert & Vanstone in [31].

The case for hyperelliptic and other non genus–one curves has been considered by Brown, Myers & Solinas [12]. This is an extension of [11] however, and is not discussed as it contains no relevant results for us here.

6.4 Compact Domain Parameters $\mathcal{V}$

In the sequel we propose a more efficient definition for publishing domain parameters than what is presented in SEC 1 from §5.2.1 and in other standards. When deploying ECC two things are usually desirable, if not required: Efficient computation of the group operation and minimal bandwidth. Whereas some of these characteristics can be improved through protocol optimisation, optimal gains can only be achieved by optimising the underlying primitives themselves.

We begin by considering how one could better represent essential components of $\mathcal{V}$ such as the field (§6.5), curve (§6.6) and order ℓ (§6.6.1). Then we propose a way to specify domain parameters which maintains efficiency and flexibility.

6.5 Defining the Field, $\mathcal{F} \in \mathcal{V}$

The definition of $\mathcal{V}$ in SEC 1 does not include non–binary extension fields (e.g. OEFs, which are particularly efficient for use with restricted devices) nor does it take into consideration the special primes, that are often suggested and used in practice. In the sequel, we give a representation for the field denoted $\mathcal{F} \in \mathcal{V}$ for large prime fields, extension fields and OEFs which are presented in §6.5.5.

When defining $\mathcal{F}$, we will assume in all non–Koblitz cases that we are using a byte–communication mode. For the Koblitz case, we assume a bit model for reasons described later in §6.5.4.

6.5.1 Compactly Representing Special Primes; Hence $\mathbb{F}_p$

Solinas in [80] listed seven families of k–bit prime numbers obtained by a direct search which he coined *generalised Mersenne numbers*. Here we list six of these families, a seventh from Smart [76] and an eighth and ninth, both of which will be useful when considering OEFs later in §6.5.5;

Type	Form		
0	$p = 2^{dw} - 3$	Crandall numbers.	[80]
1	$p = 2^{dw} - 2^{cw} - 1$	$0 < 2c \leq d$ and $\gcd(c, d) = 1$.	[80]
2	$p = \sum_{i=0}^{2d} (-1)^i 2^{(2d-i)w}$	$d \in \mathbb{N}$ and $w \not\equiv 2 \pmod 4$.	[80]
3	$p = 2^{dw} - 2^{cw} - 1$	$3d < 6c < 4d$ and $\gcd(c, d) = 1$.	[80]
4	$p = 2^{dw} - 2^{cw} + 1$	$0 < 2c < d$ and $\gcd(c, d) = 1$.	[80]
5	$p = 2^{4w} - 2^{3w} + 2^{2w} + 1$		[80]
6	$p = 2^w + c$	$1 \leq c \leq w$.	[76]
7	$p = 2^w + c$	$\lvert c \rvert \leq 2^{\lfloor w/2 \rfloor}$.	[16]
8	$p = 2^w - c$	$1 \leq c \leq w$.	

Table 6.1: Special Prime Families.

***Remark* 6.5.1.** Type–7 clearly includes Mersenne primes [65] and could represent some of the other types, albeit not as efficiently. As will be seen later, when using OEFs one usually chooses $w = 2^s$ so that w divides the word–size of a microprocessor for efficient arithmetic. For this reason we define Type–8 primes which are analogous to Type–6, but they also fit inside a given word–size.

We need to evaluate how many primes are representable in the forms given above, for a security level great enough to be suitable for the length of time of deployment. For this we use Table 6.2 given by NIST in [6] which recommends key–ranges depending on how much forward security a system requires.

Algorithmic Security Lifetime (Anni)	Year		
	$\rightarrow 2010$	$2010 \rightarrow 2030$	$2030 \rightarrow$?
k	$160 \leq k < 224$	$224 \leq k < 256$	$256 \leq k \leq 571$

Table 6.2: NIST Key–Sizes.

Definition 4.4.1 gives $k = \lfloor \lg \ell \rfloor$ for $E/\mathbb{F}_p$. From the Haße–Weil Theorem 4.1.8 when using curves with a small cofactor, one has the approximation $\ell \approx p$, thus; $\lg p \approx k$. In §6.5.5 later we will require small prime fields which fit into processor

word–sizes for OEFs, hence we will require that $\|p\| \in [4, 64]$. Combining this and Table 6.2 we present the directly computed number of primes of all forms except Type–7 in our required ranges. For the exponentially sized Type–7 case, we obtain a good approximation by using the result of Theorem 6.3.3:

Lemma 6.5.2. *Let $v_0, v_i \in \mathbb{N}$, $r = \lg e$ where e is the base of the Napierian logarithm and $B = 1.08366$. Then there exists approximately*

$$4r \sum_{v=v_0}^{v_1-1} 2^v \left(\frac{1}{2v - rB} \right) + \frac{r2^{v_1}}{2v_1 - rB}$$

Type–7 primes of k–bits in size where $2v_0 \leq k \leq 2v_1$.

Proof. The Type–7 prime is defined by $p = 2^w + c$ for a $|c| \leq 2^h$ and an $h = \lfloor w/2 \rfloor$. From Theorem 6.3.3 for a fixed w one expects the number of primes N_w which lie in the range $[x, y]_w = [2^w - 2^h, 2^w + 2^h]$ is

$$N_w = r\frac{2^h(2^{w-h}+1)}{\lg y - rB} - r\frac{2^h(2^{w-h}-1)}{\lg x - rB} \approx \frac{r2^{h+1}}{w - rB}$$

since $\lg(2^w \pm 2^h) = \lg\left(2^h(2^{w-h} \pm 1)\right) \approx h + (w - h) = w$. Assume $k_0, k_1 \in 2\mathbb{N}$ and let $k_i = 2v_i$. Now one has that the total number of Type–7 primes in the range $[k_0, k_1]$ is

$$\Big(\sum_{w=k_0}^{k_1} N_w \Big) - N_e$$

where N_e is the number of primes in the range $[2^{k_1}, 2^{k_1} + 2^h]$ since they have bit–size of $k_1 + 1$.

For a w in the range $[k_0, k_1] = [2v_0, 2v_0 + 1, 2v_0 + 2, \ldots, 2v_1]$ one has that $w = 2v + 1$ when w is odd and $w = 2v$ when even for a $v \in [v_0, v_1]$. Hence:

$$N_{2v+1} = \frac{r2^{\lfloor (2v+1)/2 \rfloor + 1}}{2v + 1 - rB} = \frac{r2^{v+1}}{2v + 1 - rB}, \qquad N_{2v} = \frac{r2^{\lfloor (2v)/2 \rfloor + 1}}{2v - rB} = \frac{r2^{v+1}}{2v - rB}.$$

By splitting and then summing over the odd and even cases for w one gets

$$\sum_{w=k_0}^{k_1} N_w = \sum_{w=k_0}^{k_1} \frac{r2^{\lfloor w/2 \rfloor + 1}}{w - rB} = r \sum_{v=v_0}^{v_1-1} \left(\frac{2^{v+1}}{2v - rB} + \frac{2^{v+1}}{2v + 1 - rB} \right) + \frac{r2^{v_1+1}}{2v_1 - rB}.$$

Since $N_e \approx r2^{k_1/2}/(k_1 - rB) = r2^{v_1}/(2v_1 - rB)$ the result now follows. □

Thus the number of primes for each type from Table 6.1 in a given range is:

Type	$4 \leq \|p\| \leq 64$	$160 \leq k \leq 224$	$224 \leq k \leq 256$	$256 \leq k \leq 572$
0	10	3	1	4
1	107	123	69	601
2	15	3	0	3
3	30	41	23	194
4	85	128	54	634
5	3	0	0	0
6	80	105	45	473
7	$\approx 2^{28.97}$	$\approx 2^{107.07}$	$\approx 2^{122.88}$	$\approx 2^{279.70}$
8	82	104	50	470
$\pi(k_0, k_1)$	$\approx 2^{58.56}$	$\approx 2^{216.73}$	$\approx 2^{248.54}$	$\approx 2^{563.37}$

Table 6.3: Prime Family Densities.

Here $\pi(k_0, k_1)$ is the expected number of general primes in the range $[k_0, k_1]$.

***Remark* 6.5.3.** The density of Type–7 primes is as expected; that is they encode just less than the square root of the expected total number of primes in a given range.

We must now choose an encoding representation for the special primes. This will allow us to measure the number of primes (hence fields) for a given type per bit–size of representation. Clearly optimising a model for primes which infrequently occur is inefficient, hence we do not consider Types 0, 2 and 5 here. We now propose models for the remaining cases:

Types 1, 3 & 4

We wish to find an efficient way of encoding primes of the form

$$p := 2^{dw} - 2^{cw} \pm 1 \tag{6.5.1}$$

for a given bit range $[k_0, k_1]$ as given in Table 6.2.

Lemma 6.5.4. *Let p, w, d and c be as above. Then one has at least three methods of specifying p by transmitting $\overline{v}$ where*

$\overline{v}$	*Bit–size for Types 1 & 4*	*Bit–size for Type–3*
$\langle p \rangle$	k_1	k_1
$\langle w, d, c \rangle$	$3\lceil \lg k_1 \rceil - 1$	$3\lceil \lg k_1 \rceil$
$\langle wd, wc \rangle$	$2\lceil \lg k_1 \rceil - 1$	$2\lceil \lg k_1 \rceil$

Proof. Trivially, sending a p in the bit range $[k_0, k_1]$ requires at most k_1–bits. Sending $\overline{v} := \langle w, d, c \rangle$ allows one to construct p and requires $\|\overline{v}\| = \lceil \lg k_1 \rceil + \lceil \lg k_1 \rceil + \lceil \lg k_1 \rceil = 3\lceil \lg k_1 \rceil$–bits for Type–3 and $\|\overline{v}\| = (3\lceil \lg k_1 \rceil - 1)$–bits for Types 1 & 4 since $2c \leq d$ here. Sending $\overline{v} := \langle wd, wc \rangle$ however is even better since $0 < cw < dw \leq k_1$. □

Definition 6.5.5. The encoding we use for a prime of the form given in equation (6.5.1) is

$$\overline{v} := \langle w(d-c), wc - 1 \rangle.$$

This is well defined since for Types 1, 3 & 4 one has $0 < wc < wd$.

Using the encoding given in Definition 6.5.5 we compute the maximum bit–sizes of $\overline{v}$ for our bit ranges. This is done exactly using the bit–size for each component of $\overline{v}$ from the prime tables of Type 1, 3 & 4 given in Appendix B.1:

		Bit–Size Range $[k_0, k_1]$			
		$[4, 64]$	$[160, 224]$	$[224, 256]$	$[256, 572]$
Type–1	$\|w(d-c)\|$	6	8	8	10
	$\|w(d-2c)\|$	6	8	8	10
	$\|wc-1\|$	5	7	7	9
Type–3	$\|w(d-c)\|$	5	7	7	9
	$\|wc-1\|$	6	8	8	9
Type–4	$\|w(d-c)\|$	6	8	8	10
	$\|w(d-2c)\|$	6	8	8	10
	$\|wc-1\|$	5	7	7	9

Table 6.4: Computed Type–1, 3 & 4 Maximal Encoded Element Bit–Sizes.

The row computing $\|w(d-2c)\|$ has been included since for Types 1 & 4, $d \geq 2c > 0$, hence $w(d-2c) \geq 0$. However, at best this saves us a bit which does not occur in our results, hence we use $w(d-c)$ in the definition of $\overline{v}$.

Types 6 & 8

We wish to find an efficient way of encoding primes of the form

$$p := 2^w \pm c \tag{6.5.2}$$

for a given bit–size range $[k_0, k_1]$ as given in Table 6.2.

Definition 6.5.6. The encoding we use for a prime of the form given in equation (6.5.2) is

$$\overline{v} := \langle w - c, c - 1 \rangle.$$

This is well defined since for Types 6 & 8 one has $1 \leq c \leq w$.

One would expect $\|\overline{v}\| = 2\lceil k_1 \rceil$–bits. Analysing Type 6 & 8 primes computed in Appendix B.1 shows this holds for our values here, as Table 6.5 demonstrates:

		Bit–Size Range $[k_0, k_1]$			
		$[4, 64]$	$[160, 224]$	$[224, 256]$	$[256, 572]$
Type–6	$\|w - c\|$	6	8	8	10
	$\|c - 1\|$	6	8	8	10
Type–8	$\|w - c\|$	6	8	8	10
	$\|c - 1\|$	6	8	8	10

Table 6.5: Computed Type–6 & 8 Maximal Encoded Element Bit–Sizes.

Type–7

We wish to find an efficient way of encoding primes of the form given equation 6.5.2 for Type–7 primes in a range $[k_0, k_1]$ as given in Table 6.2.

The only condition here on c and w is $|c| \leq 2^{\lfloor w/2 \rfloor}$. Hence, $|c| \geq 1$ or $p = 2^w$ and be composite. Similarly, we know that $w \geq k_0 > 0$, so one could transmit $w - k_0 > 0$ in lieu of w. However in the best case, both these conditions save at most one bit which is insignificant compared to the size of c. Clearly, the most 'efficient' method for transmitting p is that given in the following:

Definition 6.5.7. The encoding we use for a Type–7 prime is

$$\overline{v} := \langle w - k_0, |c|, b \rangle$$

with w, c, k_0 as before and b a bit signifying the sign of c.

Lemma 6.5.8. *To encode a Type–7 prime in the bit–size range $[k_0, k_1]$ requires*

$$\|\overline{v}\| = \lceil \lg(k_1 - k_0) \rceil + \lfloor k_1/2 \rfloor + 1 \qquad \textit{bits.}$$

Proof. By inspection of Definition 6.5.7. □

Information Rate of Representation

Here we discuss the number of fields per bit of representation. We define this is terms of an *information rate*, which is a measure of the average information being carried per symbol in a given code:

Definition 6.5.9. [66] To send some information I, let n be the number of digits used and r be the minimum necessary. Then the *information rate* (IR) of an (n, r) code with w possible codewords is $(\lg w)/n$.

When using binary, there are $w = 2^r$ possible codewords, hence; IR $= r/n$.

Definition 6.5.10. Let the information rate of our encodings of a field $(\overline{v})$ be denoted δ. Moreover;

$$\delta := \lceil \lg \#\text{fields} \rceil \, / \, \|\overline{v}\| \, .$$

Table 6.6 below the information rate δ for a given prime type:

		Bit–Size Range $[k_0.k_1]$			
		$[4, 64]$	$[160, 224]$	$[224, 256]$	$[256, 572]$
Type–1	$\|\overline{v}\|$	11	15	15	19
	#fields	107	123	69	601
	δ	0.6364	0.4667	0.4667	0.5263
Type–3	$\|\overline{v}\|$	11	15	15	18
	#fields	30	41	23	194
	δ	0.4545	0.4000	0.3333	0.4444
Type–4	$\|\overline{v}\|$	11	15	15	19
	#fields	85	128	54	634
	δ	0.6364	0.4667	0.4000	0.5263
Type–6	$\|\overline{v}\|$	12	16	16	20
	#fields	80	105	45	473
	δ	0.5833	0.4375	0.3750	0.4500
Type–7	$\|\overline{v}\|$	39	119	134	292
	#fields	$\approx 2^{28.97}$	$\approx 2^{107.07}$	$\approx 2^{122.88}$	$\approx 2^{279.70}$
	δ	0.7436	0.9076	0.9179	0.9589
Type–8	$\|\overline{v}\|$	12	16	16	20
	#fields	82	104	50	470
	δ	0.5833	0.4375	0.3750	0.4500
general	$\|\overline{v}\|$	64	224	256	572
	#fields	$\approx 2^{58.56}$	$\approx 2^{216.73}$	$\approx 2^{248.54}$	$\approx 2^{563.37}$
	δ	0.9219	0.9688	0.9727	0.9860

Table 6.6: Computed IR δ to $4d.p.$

***Remark* 6.5.11.** The result given above show how close encodings of p are to the information theoretic rate of $\delta = 1$. Clearly our encodings are far more inefficient in terms of IR then the *general* case in Table 6.6; Sending an uncompressed prime p, as an integer in the range $[k_0.k_1]$.

Clearly Type–7 primes have the best *number of curves per bit of special representation.* However, if a system only required a few fields or fields of a certain type to aid arithmetic (such as Type–1), the other encodings provide far more efficient representation than does Type–7.

Representation of Fields: $\mathcal{F}$

The information outlined in Table 6.6 allows us to decide which representation to use, based on the number of fields definable per bit of representation, for a given security parameter required.

We wish to use special forms of primes that accelerate EC group arithmetic. We also assume that a system requires at most only a few hundred fields: When a system requires more, it ought to use Type–7 primes. Since Type–1 & 4 primes have the greatest IR, we prefer these here when forming our definition of $\mathcal{F}$ for differing ranges of bit–size:

Definition 6.5.12. For ranges of bit–size $[k_0, k_1] \subseteq [160, 572]$, define the vector $\mathcal{F}_{[k_0,k_1]}$ by

$$\begin{aligned}\mathcal{F}_{[160,224]} &:= \langle b, w(d-c), wc-1\rangle \\ \mathcal{F}_{[224,256]} &:= \langle b, w(d-c), wc-1\rangle \\ \mathcal{F}_{[256,572]} &:= \langle \tau, \overline{v}_\tau\rangle\end{aligned}$$

where the bit b indicates the sign of the unit in $p = 2^{dw} - 2^{cw} + (\pm 1)^b$ for Type 1 & 4 primes. For the range $[256, 572]$, the 3–bit value τ indicates which Type $\overline{v}_\tau$ represents: Type 1,3 & 4 as was defined in Definition 6.5.5 or Type 6 & 8 as in Definition 6.5.6.

If many fields are required to be definable, one must use Type–7 primes and define

$$\mathcal{F}_{[k_0.k_1]} := \langle w - k_0, |c|, b\rangle$$

as in Definition 6.5.7.

Lemma 6.5.13. *Using Definition 6.5.12 one can show:*

$\mathcal{F}_{[k_0,k_1]}$	$\lVert\mathcal{F}_{[k_0,k_1]}\rVert$	*# of fields definable*	δ
$[160, 224]$	16	251	0.4982
$[224, 256]$	16	123	0.4339
$[256, 572]$	24	2372	0.4672

When for a given bit–size range $[k_0, k_1]$ more fields are required than definable above, one must use Type–7 primes. These require at most

$$\lceil \lg (k_1 - k_0)\rceil + \lfloor k_1/2\rfloor + 1$$

bits to represent and have an IR given in Table 6.6 above.

Proof. For $\mathcal{F}_{[160,224]}$ and $\mathcal{F}_{[224,256]}$ above, simply consult Lemma 6.5.4: b is a bit, $\|w(d-c)\| = 8$–bits and $\|wc-1\| = 7$–bits.

For $\mathcal{F}_{[256,572]}$ we encode prime Types 1, 3, 4, 6 & 8. Using their respective encodings one expects these to require no more than $10 + 10$ bits. τ needs to differentiate between these 5 cases, and so is of size 3–bits. In §6.5 we assumed we are using a byte–string communication model. Thus a single leading zero bit pads $\mathcal{F}_{[256,572]}$ to be a 3–byte string. □

6.5.2 Representing Extension Fields

It is well known that an extension field can defined by $\mathbb{F}_{p^m} := \mathbb{F}_p[X]/\langle f(X)\rangle$ where $f(X) \in \mathbb{F}_p[X]$ is the irreducible monic degree m defining polynomial. Thus one requires the pair $\{\mathbb{F}_p, f(X)\}$ to define any extension field as $\mathcal{F} \in \mathcal{V}$.

Lemma 6.5.14. *Let $\mathbb{F}_p[X]/\langle f(X)\rangle$ and $\mathbb{F}_p[Y]/\langle g(Y)\rangle$ be two different polynomial representations of the same field $\mathbb{F}_q$. Let $E/\mathbb{F}_q$ and assume an ECDLP on $E(\mathbb{F}_q)$ cannot be efficiently mapped to a DLP over a finite field $\mathbb{F}_{q^n}$.*

Then choosing a 'special' representation for $\mathbb{F}_q$ does not reduce curve security and may improve group–law performance.

Proof. It follows from §3.2.1 that any two fields of the same cardinality are isomorphic. Hence using polynomial–time algorithms (§3.3) one can compute a map between representations $\mathbb{F}_p[X]/\langle f(X)\rangle \cong \mathbb{F}_p[Y]/\langle g(Y)\rangle$.

We assume that we do not admit curves here where the ECDLP can be effectively mapped to a DLP over a finite field. Hence the security of ECC relies on the properties of the curve only and not those of the field. However, the efficiency of field arithmetic does depend on which representation one uses. □

***Remark* 6.5.15.** Lemma 6.5.14 implies that a random polynomial (hence random field representation) is in no way beneficial over choosing one with a desirable form.

We use the following Definition to aid our construction of such polynomials:

Definition 6.5.16. Let the *weight* of a polynomial $\mathrm{W}(f)$ be the number of non–zero coefficients of $f(X)$. If $\mathrm{W}(f) \leq c$ for some small $c \in \mathbb{N}$ then we say f is a *low–weight polynomial*.

When using low–weight polynomials, reduction modulo $f(X)$ is very fast as it is performed in time $O(\mathrm{W}(f) \cdot n)$ [9]. This will suit us well here, as a low weight polynomial can be specified more compactly than a random one.

6.5.3 Representing Extension Fields: Characteristic Two

To define a characteristic two field $\mathbb{F}_{2^n} := \mathbb{F}_2[X]/\langle f(X)\rangle$ one just needs to define the polynomial f. We will use the following conjecture:

Conjecture 6.5.1. *Every field extension of degree n can be represented by $\mathbb{F}_2[X]/\langle f(X)\rangle$ where*

$$\mathrm{W}(f) \leq 5.$$

Although this is conjectural, Seroussi showed in [72] that up to $n = 10000$, just over half the fields (5148) have defining polynomials of weight 3 and when not, of weight 5. Hence for the range of security parameters given in Table 6.2, we know that $\mathrm{W}(f) \leq 5$ can always be achieved.

Due to work by Frey, Galbraith, Gaudry, Heß & Smart in [29, 33] on using Weil decent as an attack, one is recommended to choose only prime extensions n (see [52]). Thus for prime n, Table 6.7 gives 60 low–weight polynomials (viz. 60 fields) contained within the cryptographically interesting range $[160, 512]$ of the form

$$f(X) = X^n + X^d + 1$$

for $\mathrm{W}(f) = 3$ and

$$f(X) = X^n + X^{d_2} + X^{d_1} + X^{d_0} + 1$$

with $n > d_2 > d_1 > d_0 > 0$ when $\mathrm{W}(f) = 5$.

(163,7,6,3)	(167,6)	(173,8,5,2)	(179,4,2,1)	(181,7,6,1)
(191,9)	(193,15)	(197,9,4,2)	(199,34)	(211,11,10,8)
(223,33)	(227,10,9,4)	(229,10,4,1)	(233,74)	(239,36)
(241,70)	(251,7,4,2)	(257,12)	(263,93)	(269,7,6,1)
(271,58)	(277,12,6,3)	(281,93)	(283,12,7,5)	(293,11,6,1)
(307,8,4,2)	(311,7,5,3)	(313,79)	(317,7,4,2)	(331,10,6,2)
(337,55)	(347,11,10,3)	(349,6,5,2)	(353,69)	(359,68)
(367,21)	(373,8,7,2)	(379,10,8,5)	(383,90)	(389,10,9,5)
(397,12,7,6)	(401,152)	(409,87)	(419,15,5,4)	(421,5,4,2)
(431,120)	(433,33)	(439,49)	(443,10,6,1)	(449,134)
(457,16)	(461,7,6,1)	(463,93)	(467,11,6,1)	(479,104)
(487,94)	(491,11,6,3)	(499,11,6,5)	(503,3)	(509,8,7,3)

Table 6.7: Values (n, d) and (n, d_2, d_1, d_0) for Low–Weight Prime Order Polynomials of Weight 3 and 5 respectively.

Using this specific polynomial representation we now have two options for how one should represent $\mathcal{F}$ for $\mathbb{F}_{2^n}$:

Compact Table Representation

If many different fields are likely to be used, then a natural solution is to store Table 6.7. We now explain how to efficiently store this data:

***Example* 6.5.17.** $n \geq 163$ is prime and hence always odd, thus set an integer $\hat{n}_i = (n - 163)/2 > 0$ which requires a maximum of $\|(509 - 163)/2\| = 8$–bits.

Each entry of $f(X) = X^n + X^d + 1$ in the $\mathrm{W}(f) = 3$ table can be encoded by

$$\langle \hat{n}, d \rangle.$$

This information requires two adjacent bytes since $\|\max\{d\}\| = \|152\| \leq 8$.

Each entry of $f(X) = X^n + X^{d_2} + X^{d_1} + X^{d_0} + 1$ in the $\mathrm{W}(f) = 5$ table is encoded by

$$\langle \hat{n}, (d_0 - 1), (d_1 - d_0), (d_2 - d_1) \rangle.$$

since $n > d_2 > d_1 > d_0 > 0$. Here one has that

$$\max \|\{\hat{n}, d_0 - 1, d_1 - d_0, d_2 - d_1\}\| = \{8, 3, 3, 4\}.$$

Hence, using padding this requires just three adjacent bytes to store an entry.

To store the table given above one requires $29(2) + 31(3) = 151$–bytes including padding. One then requires a table key, which we denote $\mathcal{H}(\hat{n})$. Thus defining $\mathcal{F} := \mathcal{H}(\hat{n})$ requires 6–bits and this is all what is needed to specify the field $\mathbb{F}_{2^n}$ using this casual compact table implementation.

This example gives an order of magnitude required to store Table 6.7 and we do not claim it is the most efficient. Using *compact prime tables* like those defined in [65] one could easily do better, but this would be warranted only if k varied across a large range which is beyond the scope of our work here.

Compact Transmission

In many cases one will not want to store this table. Here we present a compact representation for $\mathcal{F}$. To define $\mathcal{F}$ we use the analysis given in Example 6.5.17: Weight–three polynomials require 2 bytes and weight–five 3 bytes including some redundancy. We define the following:

Definition 6.5.18. Consider the binary field $\mathbb{F}_{2^n} := \mathbb{F}_2[X]/\langle f(X)\rangle$. Then the defining polynomial f is encoded by the three bytes $\{B_0, B_1, B_2\}$ as

$$\boxed{\texttt{00000000}} \parallel \boxed{d} \parallel \boxed{\hat{n}}\,,$$
$$\boxed{\texttt{000} \parallel (d_2 - d_1)} \parallel \boxed{(d_1 - d_0) \parallel (d_0 - 1) \parallel \texttt{00}} \parallel \boxed{\hat{n}}$$

where $B_0 \neq 0$ when $\mathrm{W}(f) = 5$ and values are padded where needed. Here we define $\mathcal{F} := \{B_0, B_1, B_2\}$ and trivially one has $\|\mathcal{F}\| = 24$–bits.

***Remark* 6.5.19.** With both the compact table and compact transmission methods above, devices will require additional code to encode and decode fields from and to their $\mathcal{F}$ representation. Typically this will not be much more than a few bytes, but since this is dependent on language chosen and to some degree the skill of the programmer, we do not consider this overhead here.

6.5.4 Representing Extension Fields; for Koblitz Curves

Koblitz curves will be a special case of encoding binary fields since not all degrees n are interesting now. We wish to define the field $\mathbb{F}_{2^n}$ for use with Koblitz systems from §4.2. From [16] the only suitable fields over a cryptographically interesting range are:

a	degree, n
0	$233, 239, 277, 283, 349, 409, 571$
1	$163, 283, 311, 331, 347, 359$

Table 6.8: Relevant Degrees of n for Koblitz Curves.

such that $\#E(\mathbb{F}_{2^n})/\#E(\mathbb{F}_2) = \ell$ is prime. Analogous to before, one has two approaches to represent by $\mathcal{F}$ which field we are using. First let us provide a definition for $\hat{n}$:

Definition 6.5.20. Let

$$n_i \in [163, 233, 239, 277, 283, 311, 331, 347, 349, 359, 409].$$

Then $\hat{n} = (n_i - 163)/2$ and $\hat{n} = 2^7 - 1$ when $n = 571$.

Here, $\|\max\{\hat{n}\}\| = 7$–bits. The polynomial $n = 571$ is not given in Table 6.7, thus one consults [72] to find $f := (571, 10, 5, 2)$. We now define $\mathcal{F}$:

Compact Table Representation & Transmission

Here one is only required to encode/store 12 polynomials, 4 of which are of $\mathrm{W}(f) = 3$. Here one also encodes the value of a in this representation. This will be useful in §6.8.3 later.

Definition 6.5.21. Consider the binary field $\mathbb{F}_{2^n} := \mathbb{F}_2[X]/\langle f(X)\rangle$. Then the defining polynomial f is encoded by $\mathcal{F}$ in the two bytes $\{B_0, B_1\}$ as

$$\boxed{\hat{n} \parallel a} \parallel \boxed{(d-36) \parallel 00},$$
$$\boxed{\hat{n} \parallel a} \parallel \boxed{(d_2 - d_1) \parallel (d_1 - d_0) \parallel (d_0 - 1)},$$

where since $d_0 - 1 \neq 0$, the two LSBs of the second byte indicate the $\mathrm{W}(f)$.

Lemma 6.5.22. *The encoding in Definition 6.5.21 defines (f, a) and hence $\mathbb{F}_2[X]/\langle f(X)\rangle$ for all values of n given in Table 6.5.4.*

Proof. By definition, $\|\hat{n}\| = 7$ here. Hence one can encode $\langle \hat{n}, a\rangle$ into a single byte. For $\mathrm{W}(f) = 3$, $f(X) = X^n + X^d + 1$ is given by $\langle d\rangle$. Since $n \in \{233, 239, 359, 409\}$ here, one has $36 \leq d \leq 87$. Thus $\|d - 36\| \leq \|51\| = 6$–bits.

Each $\mathrm{W}(f) = 5$ polynomial $f(X) = X^n + X^{d_2} + X^{d_1} + X^{d_0} + 1$ is specified by $\langle (d_2 - d_1), (d_1 - d_0), (d_0 - 1)\rangle$ since $n > d_2 > d_1 > d_0 > 0$. Hence one has

$$\max \|\{d_2 - d_1, d_1 - d_0, d_0 - 1\}\| = \{3, 3, 2\}$$

bits. Note that here $2 \leq d_0 \leq 5$, hence $d_0 - 1 \neq 0$. □

Corollary 6.5.23. *Definition 6.5.21 allows one to represent all binary fields $\mathbb{F}_{2^n}$ which are interesting for Koblitz ECC of size $k \in [163, 571]$ bits.*

To store the encodings of $\mathcal{F}$ in a table requires $12(2) = 24$–bytes. Let a table key $\mathcal{H}(\hat{n})$ lead one to the correct byte address. Then $\mathcal{F} = \mathcal{H}(\hat{n})$ requires 4–bits, and this is all what is needed to specify the field $\mathbb{F}_{2^n}$ using compact tables.

If one did not want to use compact tables, one would send the encoding given in Definition 6.5.21 which trivially requires two bytes of bandwidth.

6.5.5 Representing Extension Fields: Optimal Extension Fields

Optimal Extension Fields (OEFs) have become increasingly important for computationally restricted devices, such as smart–cards. In characteristic two fields, multiplication is often slower than similarly sized primes fields. This is due to a lack of single precision polynomial multiplication on microprocessors. However in prime fields, inversions are normally expensive and especially so in hardware [16]. OEFs were proposed by Bailey, Parr & Woodbury in [5, 88] to overcome these limitations.

The fundamental idea is to choose an extension field $\mathbb{F}_{p^e} := \mathbb{F}_p[X]/\langle f(X)\rangle$ such that the characteristic p fits into a machine word. This allows fast reduction modulo p and fast polynomial reduction via $f(X)$. This led Bailey et al to form the following definition:

Definition 6.5.24. An *optimal extension field* is a field

$$\mathbb{F}_{p^e} := \mathbb{F}_p[X]/\langle X^e - \omega\rangle$$

where

- p is a pseudo–Mersenne prime $p = 2^n + c$ where $|c| \leq 2^{\lfloor n/2 \rfloor}$. That is a Type–7 prime from Table 6.1.
- $X^e - \omega \in \mathbb{F}_p[X]$ is an irreducible binomial.

If $c = \pm 1$ then the field is denoted a Type–I OEF. It is of Type–II when $\omega = 2$.

The following corollary assists one in constructing OEFs:

Corollary 6.5.25. [16] *If $\omega \in \mathbb{F}_p^*$ is a primitive element and $e \mid (p-1)$, then the binomial $X^e - \omega$ is irreducible over $\mathbb{F}_p$.*

Using this, one has an efficient way of defining and searching for OEFs. After selecting a suitable prime, by Corollary 6.5.25 one can test for when

$$\omega^{(p-1)/e} \not\equiv 1 \pmod{p}$$

for candidate $\omega = 2, 3, \ldots$. When true we know $X^e - \omega$ is irreducible and can be used to define the OEF $\mathbb{F}_{p^e}$.

The computation $\omega^{(p-1)/e} \pmod{p}$ should be efficient using double–&–add or addition chains ([39], p465) and the number of trials is expected to be small, since there are $\phi(\phi(p))$ primitive roots modulo p.

For a restricted device, this computation is probably too laboursome and so here, one must transmit ω in the definition of $\mathcal{F}$. We can use a by–product of the generalised Riemann Hypothesis [54] which states that there always exists a primitive root ω such that

$$\omega < 70(\ln p)^2 < 33.7(\lg p)^2 \tag{6.5.3}$$

modulo a prime p. The details of this are beyond the scope of this thesis, but the result for the (relatively small) values of p holds well, as demonstrated by Tables 11.2 & 11.3 on pages 230 & 232 of [16] respectively. As one would expect, the values of ω in practice are much smaller than this upper bound.

In Definition 6.5.24 only Type–7 primes were defined. However, any prime may be used in the definition of an OEF as long as it fits inside a machine word. Since 8 and 16–bit word sizes are common in smart–cards and at most 64–bits for largest mainstream architectures, we considered primes of size $[4, 64]$ bits in Table 6.6.

It is reasonable to consider two encodings here; one for smart–cards using prime fields in the bit range $[4, 16]$ and one for general OEFs working with prime fields in the range $[4, 64]$ bits. This enables us to define the following:

Definition 6.5.26. For a given bit range $[4, 16]$ or $[4, 64]$ we represent an OEF $\mathbb{F}_p[X]/\langle X^e - \omega \rangle$ by

$$\mathcal{F} := \{\mathcal{F}_p, e, \omega\}$$

where ω is the smallest primitive root such that $e|(p-1)$. Here $\mathcal{F}_p$ is the representation of the field similar to that of Definition 6.5.12, that is:

$$\mathcal{F}_p := \langle b, w(d-c), wc - 1 \rangle$$

where the bit b indicates the sign of the unit in $p = 2^{dw} - 2^{cw} + (\pm 1)^b$ for Type 1 & 4 primes. If many fields are required to be definable, one must use Type–7 primes and define

$$\mathcal{F}_p := \langle w - k_0, |c|, b \rangle$$

as in Definition 6.5.7.

***Remark* 6.5.27.** When using Type–7 primes with Definition 6.5.26, one can represent any OEF.

Lemma 6.5.28. *For OEFs defined by $\mathcal{F}$, one requires*

Prime–Field Range	*Bit–size $\|\mathcal{F}\|$*	
	with Types 1 & 4	*with Type–7*
$[4, 16]$	28	34
$[4, 64]$	37	64

bits to represent OEFs of size $[160, 512]$ *bits.*

Proof. To represent $\mathcal{F}_p$ for non Type–7 primes in a bit range $[k_0, k_1]$ one requires $2\lceil \lg k_1 \rceil - 1$ bits. For Type–7 primes, one requires $\lceil \lg(k_1 - k_0) \rceil + \lfloor k_1/2 \rfloor + 1$ bits. Hence for the bit–range $[4, 16]$, one requires 7 and 13 bits respectively. Similarly in the range $[4, 64]$ one requires 12 and 39 bits respectively.

Since $k = \lfloor \lg p^d \rfloor + 1 = \lfloor d \cdot \lg p \rfloor + 1$, to encode fields up to 512–bits one must allow $\lceil \lg(511/k_0) \rceil$–bits to represent d. That is 7–bits for both bit–ranges $[4, 16]$ and $[4, 64]$. Finally, from equation (6.5.3) one has

$$\|\omega\| < 6 + 2 \lg \lg p \leq 6 + 2 \lceil \lg k_1 \rceil$$

bits. Combining these results gives the quantities defined. □

6.6 Specifying the Curve

Here we consider which curves will be representable in our definition of domain parameters, $\mathcal{V}$. We use standard curve equations, which have good features for efficient implementation.

Curves over Non–Binary Fields: Weierstraß Form

In practice one is often recommended taking $a = -3$ in the short Weierstraß form over large characteristic fields. This is because one can accelerate the group operation for this case, if field modular inversion costs significantly more than does multiplication [9]. If we are working over a field with a small characteristic $p \neq 2$, then this restriction does not in any way reduce performance since E would just be a generic curve.

This leads us to the following definition for E over non–binary fields:

Definition 6.6.1. Let the field $\mathbb{F}_q = \mathbb{F}_{p^d}$ be a power of an odd prime for $d \geq 1$. Then define the elliptic curve E to be in the short Weierstraß form

$$E/\mathbb{F}_q : y^2 = x^3 - 3x + b \tag{6.6.1}$$

with $b \in \mathbb{F}_q^*$ where $4a^3 + 27b^2 = 27(b^2 - 4) \not\equiv 0 \pmod{q}$.

Curves over Non–Binary Fields: Edwards Form

Here we consider Edwards curves presented in §4.3. In §4 of [7], Bernstein & Lange suggest that for the majority of cases, taking $c = 1$ in the Edward form accelerates group arithmetic. Once fixing c, having a small coefficient d further aids group performance. Thus we define Edwards curves of the form:

Definition 6.6.2. Let the field $\mathbb{F}_q = \mathbb{F}_{p^e}$ be a power of an odd prime for $e \geq 1$. Then define the curve E to be in Edwards form

$$E/\mathbb{F}_q : x^2 + y^2 = 1 + dx^2y^2 \tag{6.6.2}$$

with $d \in \mathbb{F}_q^* \setminus \{1\}$ where d is non–square in $\mathbb{F}_q$.

Curves over Binary Fields

For characteristic two fields we recall the form given in equation (5.4.3) where

$$E/\mathbb{F}_{2^n} : y^2 + xy = x^3 + ax + b$$

where $b \in \mathbb{F}_{2^n}^*$ and $a \in \{0, \gamma\}$ with γ a fixed element of $\mathbb{F}_{2^n}$ of trace $\mathrm{Tr}_{2^n|2}(\gamma) = 1$, [9, 75]. In §6.5.3 we restricted all binary fields to have an extension degree of prime order $n > 160$. Hence n is always odd and so one can take $\gamma = 1$ with no loss in generality. Since equation (5.4.3) represents every isomorphism class of ordinary elliptic curves over $\mathbb{F}_{2^n}$ ([9], p37), we can restrict ourselves to curves with $a \in \{0, 1\}$ which only requires a single bit. This leads us to the following definition:

Definition 6.6.3. For the field $\mathbb{F}_{2^n}$ with prime $n \geq 160$, define E to be of the form

$$E/\mathbb{F}_{2^n} : y^2 + xy = x^3 + ax + b \tag{6.6.3}$$

where $a \in \{0, 1\}$ and $b \in \mathbb{F}_{2^n}^*$.

We now discuss other criteria for the construction of suitable curves $E/\mathbb{F}_q$.

6.6.1 The Orders of the Curve: $\#E$ and ℓ

All known standards require that the order of the curve is divisible by a large prime ℓ. This is because ECDLP cryptography operates in the subgroup $\langle G \rangle \subseteq E(\mathbb{F}_q)$ of order ℓ. Since the security of a curve k is proportional to ℓ and not directly to $\#E$, curves with large cofactors $c = \#E/\ell$ are *over–specified*:

Definition 6.6.4. The *minimal cofactor* $c = \#E(\mathbb{F}_q)/\ell$ of an elliptic curve $E/\mathbb{F}_q$ is defined by

$$c := \begin{cases} 1 & \text{when } \mathrm{char}(\mathbb{F}_p) \neq 2, \\ \#E(\mathbb{F}_p) & \text{when } \mathrm{char}(\mathbb{F}_p) \neq 2 \text{ and } E/\mathbb{F}_p, \\ 2(2-a) & \text{when } q = 2^n, \\ 8 & \text{when using Edwards curves.} \end{cases} \tag{6.6.4}$$

Thus in all cases, the minimal cofactor is simply the smallest cofactor possible.

Definition 6.6.5. Let k be a fixed security parameter of $E/\mathbb{F}_q$, c the cofactor of the curve and c_m be the minimal cofactor from Definition 6.6.4. Then having a cofactor $c > c_m$ increases bandwidth with no increase in security. Here we say the curve $E/\mathbb{F}_q$ is *over–specified* by $\lceil \lg c \rceil - \lceil \lg c_m \rceil$ bits.

***Remark* 6.6.6.** When using subfield curves over OEFs, one has a point redundancy of $\lceil \lg \#E(\mathbb{F}_p) \rceil$ bits which could be quite significant. For this reason, we do not consider the use of subfield curves defined over OEFs in the sequel.

Hence in many ways it is desirable to make the cofactor minimal, as no additional security is gained by increasing c at the expense of both arithmetic and storage efficiency. Thus we restrict all curves E to have a minimal cofactor.

Corollary 6.6.7. *If one defines $E/\mathbb{F}_q$ to always have a minimal cofactor, then one does not require c to be defined as a domain parameter, as its value is implicit from the definition of the curve and field.*

Hence, in our definition of domain parameters that follows in §6.6.5, we construct $\mathcal{V}$ without explicitly defining c.

6.6.2 Choosing the Coefficient b for Weierstraß Forms

We wish to define elliptic curves in line with Definitions 6.6.1 & 6.6.3. Normally when generating suitable curves, a system or user decides whether it needs to be provably random. If this is the case, one fixes the coefficient a and uses a seed and suitably sized hash function to generate a value

$$b = \mathcal{H}_k(\texttt{seed}) \pmod{q}.$$

One would then test if the resulting curve had the desired properties, and repeat the curve selection process if not. The coefficient b requires $\lceil \lg q \rceil$–bits to represent here, or, the output size of $\mathcal{H}_k$ which might be $< \lceil \lg q \rceil$.

If a certificate is not required, one can do better: Instead of drawing random values $b \in \mathbb{F}_q^*$, there is no known risk in restricting b to be a 'small' element of $\mathbb{F}_q^*$. We now clarify this idea further.

Definition 6.6.8. For $\mathrm{char}(\mathbb{F}_q) \neq 2$, let $r \in \mathbb{N}$ such that $r \ll \lceil \lg q \rceil$. Then $b \in \mathbb{F}_q^*$ is denoted an *r–small element* if

$$b \leq 2^r - 1$$

when recognised as an integer modulo q.

Definition 6.6.9. For $\mathbb{F}_{2^n}$, let $r \in \mathbb{N}$ such that $r \ll n$. Then $b \in \mathbb{F}_{2^n}^*$ is denoted an *r–small element* if

$$\deg(b) \leq r - 1.$$

Heuristic 6.6.10. Let $E/\mathbb{F}_q$ be an elliptic curve represented via the short Weierstraß form. Then in general, an E constructed by an r–small coefficient is no weaker than a curve constructed with a random element $b \in_R \mathbb{F}_q^*$.

In our definition of $\mathcal{V}$ we will generate curves with r–small coefficients: That is, fix a as defined in Definitions 6.6.1 & 6.6.3 and generate r–small elements $b \in \mathbb{F}_q^*$ until one has curves with the desired properties. To find an empirical bound for the size of b, we use the work of Galbraith & McKee from [30]:

6.6.3 Probability of Occurring Orders of $\#E$

Galbraith & McKee in [30] investigate the probability that a random curve over a finite field has prime or near prime order. They give the following conjectures:

Conjecture 6.6.1 (Conjecture A, [30]). *Let P_1 be the probability that a number within $2\sqrt{p}$ of $p+1$ is prime. Then the expected probability that $E/\mathbb{F}_p$ has a prime number of points is asymptotic to $c_p P_1$ as $p \to \infty$, where $0.44 \leq c_p \leq 0.62$ and P_1 can be approximated by*

$$\frac{1}{4\sqrt{p}} \int_{p+1-2\sqrt{p}}^{p+1+2\sqrt{p}} \frac{\mathrm{d}t}{\ln t} \approx \frac{1}{\ln p} = \frac{\lg e}{\lg p}.$$

Remarks

1. Note that the starting point for the construction of Conjecture A is the Kronecker/Hurwitz class number: Given $|t| < 2\sqrt{p}$, the probability $E/\mathbb{F}_p$ has exactly $p+1-t$ points is

   $$\mathrm{H}(t^2 - 4p)/2p.$$

   This formula still holds when one has $q = p^m$ for some $m > 1$, where one now has $\mathrm{H}(t^2 - 4q)/2q$. Thus we can use this result when looking for prime order curves over extension fields $\mathbb{F}_q \cong \mathbb{F}_{p^m}$ also.

2. As noted in [30], if all numbers of points in the range $p+1-2\sqrt{p}$ to $p+1+2\sqrt{p}$ where equally likely, one would have a $c_p = 1$. The conjectured result of $0.44 \leq c_p \leq 0.62$ indicates the phenomenon that prime orders of E are disfavoured.

Conjecture 6.6.2. ([30], p12) *Let $t = 1$ or 2, $q = 2^n$ and P_t be the probability that an even number within $2\sqrt{q}$ of $q+1$ is 2^t times a prime. Then the expected probability that $E/\mathbb{F}_q$ has order 2^t times a prime is asymptotic to $c_q P_t$ as $n \to \infty$ where*

$$c_q = \prod_{l>2} \left(1 - \frac{1}{(l-1)^2}\right) \prod_{l \mid 2^n - 1} \left(1 + \frac{1}{(l+1)(l-2)}\right).$$

Remarks

1. P_t can be approximated via

   $$\frac{1}{4 \cdot 2^{t-1}\sqrt{q}} \int_{(q+1-2\sqrt{q})/2^t}^{(q+1+2\sqrt{q})/2^t} \frac{\mathrm{d}t}{\ln t} \approx \frac{1}{2^{t-1}(\ln q - \ln 2^t)} = \frac{2^{1-t} \lg e}{n-t}.$$

2. The product

   $$\prod_{l>2} \left(1 - \frac{1}{(l-1)^2}\right) \approx 0.6601618158\ldots$$

   is the well known Hardy–Littlewood twin–primes constant Π_2 [89].

Not enough is known about primes in short intervals to say if these these conjectures are correct. However empirical results hold with an astonishing degree of accuracy as demonstrated in [30] and by our results in Examples 6.6.13 & 6.6.16. Since the order of E is essentially independent whether one varies a or b, here we can fix a and vary b until we have a desired curve. This allows us to form the following:

For Prime–Order Curves over $\mathbb{F}_q$

Theorem 6.6.11. *Let $E/\mathbb{F}_q$ be non–binary whose coefficients are $a = -3$ and $b \in \mathbb{F}_q^*$ such that b is r–small. Then using Conjecture 6.6.1 one can define an expected*

$$0.63 \cdot 2^r \cdot k^{-1}$$

prime order curves E with security parameter $k = \lfloor \lg q \rfloor$.

Proof. For curves $E/\mathbb{F}_q$ with minimal cofactor $c = 1$, Conjecture 6.6.1 implies for a fixed $a \in \mathbb{F}_p$ and random $b \in_R \mathbb{F}_q^*$, the probability $\mathbb{P}_q$ that E is of prime order is bound by

$$0.44 \cdot \frac{\lg e}{\lg q} \leq \mathbb{P}_q \leq 0.62 \cdot \frac{\lg e}{\lg q}.$$

Taking $b = 1, 2, 3, \ldots$ where $4a^3 + 27b^2 \not\equiv 0 \pmod{q}$, one gets an upper bound for the expected number of trials to find a prime order curve to be

$$b = \left\lceil (\min\{\mathbb{P}\})^{-1} \right\rceil = \lceil 1.5753 \cdot \lg q \rceil < 1.581k$$

for $k \geq 160$.

Thus, if one selects only r–small b from $\mathbb{F}_q^*$, one can expect out of the $2^r - 1$ possible choices for b that

$$\approx \frac{2^r}{1.581k}$$

curves of prime order exist. The result now follows. □

Corollary 6.6.12. *Let $\mathbb{F}_q$ be non–binary and let $a = -3$ and b be r–small. Then in order to define an expected 2^h elliptic curves $E_i : y^2 = x^3 - 3x + b$ one requires an*

$$r \geq \lg(1.581 \cdot 2^h \cdot k)$$

which implies

$$\|b\|_h \leq \lceil h + \lg k + 0.661 \rceil .$$

We now present an example of this in action.

***Example* 6.6.13.** Let $p = 2^{192} - 2^{64} - 1$ and fix $a = -3$ in the short Weierstraß form of $E_i : y^2 = x^3 - 3x + b$. Then suitable b which yield a prime order curve are:

h	b	trace: $p + 1 - \#E$	$\lg b$	$\|b\|_h$
0	446	136061485252480925449033158677	8.801	$\lceil 8.246 \rceil$
1	537	59802929316165084957215634253	9.069	$\lceil 9.246 \rceil$
$\lg 3$	658	-60958393595616787102755480763	9.362	$\lceil 9.831 \rceil$
2	1013	13239429162803941121486014513	9.984	$\lceil 10.246 \rceil$
$\lg 5$	1388	12168542991176887019335499309	10.439	$\lceil 10.568 \rceil$
$\lg 6$	2111	-40255677575921460661737995251	11.044	$\lceil 10.831 \rceil$
$\lg 7$	2366	-49015247589235182139725640813	11.208	$\lceil 11.053 \rceil$
3	2606	82545172752563107569454411387	11.348	$\lceil 11.246 \rceil$
$\lg 9$	2838	12719827397689782915666995036 9	11.471	$\lceil 11.412 \rceil$
$\lg 10$	3032	57195467430761463244394803949	11.566	$\lceil 11.568 \rceil$
$\lg 11$	3511	22807430298097133927723529551	11.778	$\lceil 11.705 \rceil$
$\lg 12$	3526	11305008214787630978751267742 9	11.784	$\lceil 11.831 \rceil$
$\lg 13$	3664	53986358582377581168857068861	11.839	$\lceil 11.946 \rceil$

Here one can see the difference between the actual size of b and the expected size of b holds very well and improves as h grows. This supports Corollary 6.6.12.

For Curves over $\mathbb{F}_{2^n}$

In this section we consider non–Koblitz curves $E/\mathbb{F}_q$, since b is fixed for these cases. We follow an analogous approach as we did for curves over non–binary fields here, however since we only consider fields of certain prime degree, one can compute c_q exactly. Following Table 6.7 and using Conjecture 6.6.2 where

$$c_q = \Pi_2 \cdot \prod_{l \mid 2^n - 1} \left(1 + \frac{1}{(l+1)(l-2)}\right)$$

with Π_2 being the Hardy–Littlewood twin-prime constant one has

$$\Pi_2 \leq c_q \leq 0.660167$$

as shown in Table B.21 in Appendix B.2. This tight bound is expected since no known Mersenne number $M_n = 2^n - 1$ for a prime n, has a square as a root [35]. Thus the factors l of M_n are generally large. This yields the following:

Theorem 6.6.14. *Let $E/\mathbb{F}_{2^n}$ with prime $n > 160$, coefficients $a \in \{0, 1\}$ and $b \in \mathbb{F}_{2^n}^*$ which is r–small. Then using Conjecture 6.6.2 one can define an expected*

$$0.89 \cdot n^{-1} \cdot 2^{a+r}$$

curves E with minimal cofactor from Definition 6.6.4.

Proof. Following the proof of Lemma 6.6.11: Conjecture 6.6.2 and Table B.21 implies for a fixed $a \in \mathbb{F}_{2^n}$ and random $b \in \mathbb{F}_{2^n}^*$ the probability $\mathbb{P}_n$ that E is of order 2^t times a prime is lower bounded by

$$\Pi_2 \cdot \frac{2^{1-t} \lg e}{n - t} \leq \mathbb{P}_n \qquad \text{as } n \to \infty.$$

Taking $b = 1, z, 1 + z, \ldots$ where $f(z) = 0$, the expected number of trials to find a curve of the desired order is $\approx \mathbb{P}_n^{-1}$. Hence, if one selects only r–small b from $\mathbb{F}_{2^n}^*$, one expects out of the $2^r - 1$ possible choices for b that approximately

$$(2^r - 1) \cdot \Pi_2 \cdot \frac{2^{1-t} \lg e}{n - t} \approx 1.79 \cdot \frac{2^{r+1-t}}{n}$$

curves of desired order exist. Noting that here $t = (2 - \mathrm{Tr}_{2^n|2}(a))$ the result now follows. □

Corollary 6.6.15. *For $\mathbb{F}_{2^n}$ with a prime $n > 160$, fix an $a = \{0, 1\}$ and let b be r–small. Then in order to define an expected 2^h elliptic curves $E_i : y^2 + xy = x^3 + ax + b$ one needs an*

$$r \geq \lg(1.13 \cdot 2^{h-a} \cdot n)$$

which implies

$$\|b\|_h \leq \lceil h + \lg n + 0.18 - a \rceil .$$

***Example* 6.6.16.** Let $n = 281$ and let $f(X) = X^{281} + X^{93} + 1$ as was defined in Table 6.7. Canonically construct $\mathbb{F}_{2^n} := \mathbb{F}_2[X]/\langle f(X) \rangle$ and define $E_i : y^2 + xy = x^3 + ax + b$ where $a \in \mathbb{F}_2$. Then suitable b for an $f(z) = 0$ which yield curves or maximal order are:

For $a = 0$:

h	b	$\deg(b)$	$\|b\|_h$
0	$z^9 + z^4 + z^2 + z + 1$	10	$\lceil 8.314 \rceil$
1	$z^{10} + z^6 + z^4 + z^3 + z^2 + z + 1$	11	$\lceil 9.314 \rceil$
$\lg 3$	$z^{10} + z^7 + z^6 + z^3 + z^2 + 1$	11	$\lceil 9.899 \rceil$
2	$z^{10} + z^9 + z^7 + z^6 + z^5 + z^2 + z + 1$	11	$\lceil 10.314 \rceil$
$\lg 5$	$z^{11} + z^9 + z^5 + z^3 + z + 1$	12	$\lceil 10.636 \rceil$
$\lg 6$	$z^{11} + z^9 + z^8 + z^6 + z^4 + z^3 + z + 1$	12	$\lceil 10.899 \rceil$
$\lg 7$	$z^{11} + z^{10} + z^7 + z^6 + z^5 + z^4 + z^3 + z^2 + z + 1$	12	$\lceil 11.122 \rceil$
3	$z^{11} + z^{10} + z^8 + z^6 + z^5 + z^4 + z^3 + z + 1$	12	$\lceil 11.314 \rceil$
...	...	...	...
4	$z^{12} + z^{10} + z + 1$	13	$\lceil 12.314 \rceil$
5	$z^{13} + z^9 + z^8 + z^5 + z^4 + z^2 + 1$	14	$\lceil 13.314 \rceil$
6	$z^{14} + z^{11} + z^9 + z^8 + z^7 + z^3 + z^2 + 1$	15	$\lceil 14.314 \rceil$
7	$z^{15} + z^{13} + z^{10} + z^5 + z^3 + z^2 + 1$	16	$\lceil 15.314 \rceil$

Table 6.9: Example Maximal Order Curves $E/\mathbb{F}_{2^{281}}$ for $a = 0$.

For $a = 1$:

h	b	$\deg(b)$	$\|b\|_h$
0	$z^8 + z^5 + z^4 + z^3 + z^2 + z$	9	$\lceil 7.314 \rceil$
1	$z^9 + z^7 + z^5 + z^4$	10	$\lceil 8.314 \rceil$
2	$z^{10} + z^7 + z^6 + z^4 + z + 1$	11	$\lceil 9.3144 \rceil$
3	$z^{11} + z^7 + z^5 + z^3 + z^2$	12	$\lceil 10.314 \rceil$
4	$z^{11} + z^{10} + z^6 + z^5 + z^4 + z^3 + z^2 + 1$	12	$\lceil 11.314 \rceil$
5	$z^{12} + z^{10} + z^9 + z^6 + z^3 + z + 1$	13	$\lceil 12.314 \rceil$
6	$z^{13} + z^{11} + z^9 + z^7 + z^3 + z^2$	14	$\lceil 13.314 \rceil$
7	$z^{14} + z^{11} + z^9 + z^5 + z^4 + z$	15	$\lceil 14.312 \rceil$

Table 6.10: Example Maximal Order Curves $E/\mathbb{F}_{2^{281}}$ for $a = 1$.

Here one can see the difference between the actual and the expected size of b holds very well, improving as h grows. This supports Corollary 6.6.15.

6.6.4 Choosing the Coefficient d for Edwards Forms

As mentioned in §6.6, one desires a Edwards curve

$$E/\mathbb{F}_q : x^2 + y^2 = 1 + dx^2y^2 \tag{6.6.5}$$

with a small non–square $d \in \mathbb{F}_q^* \setminus \{1\}$ as it aids arithmetic. This suits us very well here, and we give an overview of what one would do for this case:

Edwards curves over finite fields always contain a point of order two and a point of order four; viz. Theorem 4.3.3. Hence their minimal cofactor is 8 as was given in Definition 6.6.4. We wish to search for Edwards curves $E/\mathbb{F}_q$ of order 8ℓ for some prime ℓ. Point counting routines are not yet known for Edwards curves, but computing the corresponding Weierstraß form (when it exists) is essentially free, allowing us to use the polynomial–time ones of Schoof, [70]. Hence one proceeds analogously to §6.6.2 to find suitable *minimal Edwards curves*:

Algorithm 6.6.1 (Constructing Minimal Edwards Curves).
INPUT: $\mathbb{F}_q$ and a parameter $s \in \mathbb{N}$;
OUTPUT: An expected 2^s coefficients d_i defining minimal Edwards curves or fail.

1. set an array $\overline{d} \leftarrow$ null and $j \leftarrow 0$;
2. for $(d = 2; d < q; i$++$)\{$
3. if IsNonSquare$(d)\{$
4. set $E \leftarrow x^2 + y^2 = 1 + dx^2y^2$;
5. if IsomorphismExistsToAWeirstraßForm$(E)\{$
6. construct isomorphism: $\phi : E/\mathbb{F}_q \to E'/\mathbb{F}_{q^e}$;
7. compute Weierstraß form: $E' = \phi(E)$;
8. if IsPrime$(\#E'(\mathbb{F}_{q^e})/8)\{$;
9. set $\overline{d}[j$++$] \leftarrow d$;
10. if $j \geq 2^s$ return $\overline{d}$;
11. $\}$
12. $\}$
13. $\}$
14. $\}$
15. return fail;

For a given s we need to estimate the expected size of d. Clearly one expects a Weierstraß curve to have order 8ℓ to be asymptotic to c_eP_e as $q \to \infty$ where $c_e \in (0,1) \subset \mathbb{R}$ and P_e can be approximated by

$$\frac{1}{4 \cdot 2^2\sqrt{q}} \int_{(q+1-2\sqrt{q})/8}^{(q+1+2\sqrt{q})/8} \frac{\mathrm{d}t}{\ln t} \approx \frac{1}{2^2(\ln q - \ln 8)} = \frac{\lg e}{8(\lg q - 3)}.$$

We do not compute the specific c_e here as was done by Galbraith & McKee in Conjectures 6.6.1 & 6.6.1 for the Weierstraß form. We merely note that one has a similar result to the Weierstraß form whereby one can define an r–small d sufficiently big enough to contain 2^s choices for $\mathcal{V}$.

6.6.5 Summary

We now condense the preceding sections into our formal definition for $E \in \mathcal{V}$:

Definition 6.6.17. Let $\mathbb{F}_q := \mathbb{F}_{p^d}$ be a power of an odd prime with $d \geq 1$. Then represent $E/\mathbb{F}_q : y^2 = x^3 - 3x + b$ by

$$\langle b, \hat{t} \rangle$$

where $b \in \mathbb{F}_q^*$ and $\hat{t}$ is the modified trace of $E/\mathbb{F}_q$ as given in Definition 5.4.1 (equivalently for subfield curves, Definition 5.6.4). Implicitly the curve here has minimal cofactor as given in Definition 6.6.4.

Corollary 6.6.18. *For an expected 2^h definable curves, one has*

$$\left\| \langle b, \hat{t} \rangle \right\| = \lceil h + \lg k + 0.661 \rceil + \lceil (k+1)/2 \rceil + 1.$$

Proof. From Corollary 6.6.12 one has that $\|b\| \leq \lceil h + \lg k + 0.661 \rceil$. From Lemma 5.4.2 one has that $\left\|\hat{t}\right\| = \lceil ((17/16)k + 1)/2 \rceil + 1$ however one does not have $1 \leq c \leq 2^{k/16}$ but $c = 1$, hence $\left\|\hat{t}\right\| = \lceil (k+1)/2 \rceil + 1$ here. The result now follows. □

Definition 6.6.19. Let $\mathbb{F}_{2^n}$ have prime $n \geq 160$. Then represent $E/\mathbb{F}_{2^n}$: $y^2 + xy = x^3 + ax + b$ by

$$\langle a, b, \hat{t} \rangle$$

where $a \in \{0, 1\}$, $b \in \mathbb{F}_{2^n}^*$ and $\hat{t}$ is the modified trace of $E/\mathbb{F}_{2^n}$ as given in Definition 5.4.4. Implicitly the curve here has minimal cofactor as given in Definition 6.6.4.

Corollary 6.6.20. *For an expected 2^h definable curves, one has*

$$\left\|\langle a, b, \hat{t} \rangle\right\| = \lceil h + \lg(k + 3 - a) + 0.18 \rceil + \lceil (k + 1)/2 \rceil + 1.$$

Proof. From Corollary 6.6.15 one has that $\|b\| \leq \lceil h + \lg n + 0.18 - a \rceil$. From Lemma 5.4.5 and Corollary 6.6.18 one has that $\left\|\hat{t}\right\| = \lceil (k + 1)/2 \rceil + \mathrm{Tr}_{2^n|2}(a) = \lceil (k + 1)/2 \rceil + a$ here. Finally note that in the worst case $k = (n - 1) - \lg c = n + a - 3$. The result now follows. □

Definition 6.6.21. Let $\mathbb{F}_{2^n}$ have prime $n \geq 160$. Then represent Koblitz curves $E/\mathbb{F}_{2^n} : y^2 + xy = x^3 + ax + 1$ by

$$\langle a \rangle$$

where $a \in \{0, 1\}$. Implicitly the curve here has cofactor $c = 2(2 - a)$ and the order is either known or trivial to compute as was given in Lemma 5.6.7.

Corollary 6.6.22. *The value a is already included in the definition of the field representation $\mathcal{F}$: Definition 6.5.21.*

6.7 Specifying Base Points

A generating point P of order ℓ is required to be defined in $\mathcal{V}$, so that users may construct their public/private key pairs $(Q = [\varphi]P, \varphi)$. It is often recommended that one chooses this base point at random by selecting an $x_i \in_R \mathbb{F}_q$ and testing whether or not $x_i^3 + ax_i + b$ is a quadratic residue modulo $p = \mathrm{char}(\mathbb{F}_q)$ [2, 9, 14]. A similar method is used for curves over $\mathbb{F}_{2^n}$ and both have a probability of success in defining a valid curve point of a $1/2$ for each trial. Once a valid point $P \in E(\mathbb{F}_q)$ has been found, it is checked if it is of the correct order ℓ; if not the implementer decides whether they draw another random value/seed for x_i, or whether they indicate to the receiver by an additional bit that they need to compute $P' = [c]P$, ensuring P' has the correct order ℓ.

Here instead we restrict users to only define base points using r–small abscissæ x_i. If these points $P \in E(\mathbb{F}_q)$ are of the correct order ℓ, and the order is large enough to avoid pre–computation (which is implicit), they trivially admit no weakness in an ECDLP. We present this now:

Lemma 6.7.1. *Let $x_i \in \mathbb{F}_q$ such that x_i is r–small. Assume one needs to define 2^s base points $P = (x_i, y_i)$, where $r > s - 1$, is of maximal order $\ell = \#E/c$ with minimal $y_i \leq (q - 1)/2$. Then one has a probability for success of*

$$(1 - 2^{2-2^r})/c2^s.$$

Proof. Consider $E/\mathbb{F}_p$: One successively tests whether $x_i = 1, 2, 3, \ldots \leq 2^r - 1$ is a quadratic residue modulo p which fails with the probability $1/2$. The abscissæ x_i are successive, however one may view $x_i^3 + ax_i + b$ as a quasi–random value modulo p. This gives the probability of finding a pair of points of $E(\mathbb{F}_p)$ as $\mathbb{P}_P = 1 - (\frac{1}{2})^n$ for n trials. Thus one has a probability of

$$\mathbb{P}_h = 2^{-h}(1 - 2^{2-2^r})$$

for finding 2^s such points since $x_i \neq 0$, $n \leq 2^r - 2$.

Now that one has valid points, the probability that they are of order ℓ is $\ell/\#E = c^{-1}$. Once points $P = (x_i, \pm y_i)$ have been found, we choose the smallest ordinal from $\pm y_i$ since if $-P$ is of order ℓ so is P since $\mathcal{O} = [\ell](-P) = [-1 \cdot \ell]P \Leftrightarrow [\ell]P = \mathcal{O}$. An analogous argument exists for curves over binary fields and the result now follows. □

We wish to be able to define points with a high degree of confidence. For a $c = 1$ and varying r, one finds that the probability for each trial of defining a 3–small P as $63/64$, and a 4–small P as $16383/16384$. This leads us to:

Corollary 6.7.2. *If a system requires 2^s base points P_i, then with an overwhelming confidence of* 0.9999 *one can express such a point using r–small elements where $r = 4 + \lceil s + \lg c \rceil$. Such elements clearly require only r–bits to represent.*

Corollary 6.7.3. *Should only one base point P be sought, then with an overwhelming confidence of* 0.9999 *one only requires an:*

$$r := \begin{cases} 4 & \text{when } \mathrm{char}(\mathbb{F}_p) \neq 2, \\ 5 + \lceil \lg p \rceil & \text{when } \mathrm{char}(\mathbb{F}_p) \neq 2 \text{ and } E/\mathbb{F}_p, \\ 5 - a & \text{when } q = 2^n \text{ for the coefficient } a \text{ of } E. \end{cases} \tag{6.7.1}$$

Here one assumes the cofactor is minimal from Definition 6.6.4 and for the binary cases, we are using Seroussi's method from §4.1.3.

6.8 Definition and Generation of Multi–Curve CDPs

Here we combine the work undertaken in the previous sections, and present the result.

6.8.1 For Prime Fields $\mathbb{F}_p$

Here we present the formal definition for our domain parameters $\mathcal{V}$, suitable for ECC for multi–curve environments over prime fields:

Definition 6.8.1. For a given bit–size $k \in [k_0, k_1]$, the domain parameters for a 2^h multi–curve environment where $E/\mathbb{F}_p$ are:

$$\mathcal{V} := \mathcal{F} \cup (b, \hat{t}, xP).$$

Here one has:

- $\mathcal{F}$ describing the field as was given in Definition 6.5.12;

- $\langle b, \hat{t} \rangle$ is an r–small coefficient and the modified trace defining the curve $E/\mathbb{F}_p$ as was given in Definition 6.6.17 such that an expected 2^h curves are definable;
- $\hat{t}$ is the modified trace from Definition 5.4.1;
- and xP is an r–small element from Corollary 6.7.2 which defines the base point $P = (xP, yP)$ for a $yP \leq (p-1)/2$.

Using this and the fact that $E : y^2 = x^3 - 3x + b$ has a minimal cofactor $c = 1$ as in Definition 6.6.4, one is able to define valid parameters for ECC.

Corollary 6.8.2. *Once $\mathcal{F}$ is established, the specific security parameter k is known. Hence for our $\mathcal{V}$ defined above one has*

$$\left\| b, \hat{t}, xP \right\| = \lceil h + \lg k + 0.661 \rceil + \lceil (k+1)/2 \rceil + \lceil s \rceil + 5,$$

where one requires 2^s base points P.

Depending on how many fields a system needs to define, using Lemma 6.5.13 we have:

$[k_0, k_1]$	*# of fields*	$\|\mathcal{V}\|$
$[160, 224]$	251	$\lceil h + \lg k + 0.661 \rceil + \lceil (k+1)/2 \rceil + \lceil s \rceil + 21$
$[160, 224]$	$\approx 2^{107.07}$	$\lceil h + \lg k + 0.661 \rceil + \lceil (k+1)/2 \rceil + \lceil s \rceil + 124$
$[224, 256]$	123	$\lceil h + \lg k + 0.661 \rceil + \lceil (k+1)/2 \rceil + \lceil s \rceil + 21$
$[224, 256]$	$\approx 2^{122.88}$	$\lceil h + \lg k + 0.661 \rceil + \lceil (k+1)/2 \rceil + \lceil s \rceil + 139$
$[256, 572]$	2372	$\lceil h + \lg k + 0.661 \rceil + \lceil (k+1)/2 \rceil + \lceil s \rceil + 29$
$[256, 572]$	$\approx 2^{279.70}$	$\lceil h + \lg k + 0.661 \rceil + \lceil (k+1)/2 \rceil + \lceil s \rceil + 297$

Moreover, by direct calculation for all $k \in [k_0, k_1]$ one has that

$[k_0, k_1]$	$\|\mathcal{V}\|$ for SEC 1	$\|\mathcal{V}_m\|$	$\|\mathcal{V}_f\|$
$[160, 224]$	$\leq 5.393k$	$\leq 1.098k + \delta$	$\leq 0.638k + \delta$
$[224, 256]$	$\leq 5.469k$	$\leq 1.082k + \delta$	$\leq 0.621k + \delta$
$[256, 572]$	$\leq 5.385k$	$\leq 1.038k + \delta$	$\leq 0.570k + \delta$

where $\mathcal{V}_f$ is when requiring few fields to be definable and $\mathcal{V}_m$ many. Here $\delta = \lceil h \rceil + \lceil s \rceil$.

We now present an example of this in action and compare it against SEC 1 domain parameters from Definition 5.2.2:

***Example* 6.8.3.** Let $E : y^2 = x^3 - 3x + b$ be as in Example 6.6.13 where $p = 2^{192} - 2^{64} - 1$. Assume as is usual that we only require one base point P, hence $s = 0$. Set $h = 8$; then one can represent a $\mathcal{V}$ for an expected 256 multi–curve environment using Definition 6.8.1 in

$$\lceil h + \lg k + 0.661 \rceil + \lceil (k+1)/2 \rceil + \lceil s \rceil + 21 = 127\text{–bits}.$$

From Lemma 5.2.4, one has the size for SEC 1 domain parameters to be

$$32 \lceil ((17/16)k + 1)/8 \rceil + 8 \lceil (k+1)/8 \rceil + 8 \lceil k/128 \rceil = 1056\text{–bits}.$$

when using point compression.

***Remark* 6.8.4.** For OEFs, one has the analogous result where one uses $\mathcal{F}$ from Definition 6.5.26 to define our $\mathcal{V}$. This results in a slightly larger $\|\mathcal{V}\|$ following the result of Lemma 6.5.28.

6.8.2 For Binary Fields $\mathbb{F}_{2^n}$

Here we present the formal definition for our domain parameters $\mathcal{V}$, suitable for ECC for multi–curve environments over binary fields:

Definition 6.8.5. The domain parameters for a 2^h multi–curve environment where $E/\mathbb{F}_{2^n}$ are:

$$\mathcal{V} := \mathcal{F} \cup (a, b, \hat{t}, xP).$$

Here one has:

- $\mathcal{F}$ describing the field as was given in Example 6.5.17 and Definition 6.5.18;
- $\langle a, b, \hat{t} \rangle$ has a bit a, an r–small coefficient b and the modified trace defining the curve $E/\mathbb{F}_{2^n}$ as was given in Definition 6.6.19 such that an expected 2^h curves are definable;
- $\hat{t}$ is the modified trace from Definition 5.4.4;
- xP is an r–small element from Corollary 6.7.2 which defines the base point $P = (xP, yP)$ with $yP := \min\{\pm yP\}$ computed via the Weierstraß form.

Using the fact that $E : y^2 + xy = x^3 + ax + b$ has a minimal cofactor $c = 2(2 - a)$ as in Definition 6.6.4, one is now able to define valid parameters for ECC.

Corollary 6.8.6. *Once $\mathcal{F}$ is established, the specific security parameter k is known. Hence for our $\mathcal{V}$ defined above one has*

$$\|\mathcal{V}\| = \lceil h + \lg(k + 3 - a) + 0.18 \rceil + \lceil (k + 1)/2 \rceil + \lceil s \rceil + 15 - a,$$

when using compact tables and

$$\|\mathcal{V}\| = \lceil h + \lg(k + 3 - a) + 0.18 \rceil + \lceil (k + 1)/2 \rceil + \lceil s \rceil + 31 - a,$$

without where one requires 2^s base points P.

By direct calculation for all selected $k \in [162, 510]$ one has that

	$\|\mathcal{V}\|$ for SEC 1	$\|\mathcal{V}\|$	$\|\mathcal{V}_t\|$
k	$\leq 5.381k$	$\leq 0.583k + \delta$	$\leq 0.551k + \delta$

where $\mathcal{V}_t$ is when using compact tables and $\mathcal{V}$ when not where $\delta = \lceil h \rceil + \lceil s \rceil$.

6.8.3 For Koblitz Curves $E/\mathbb{F}_{2^n}$

Here we present the formal definition for our domain parameters $\mathcal{V}$, suitable for ECC for multi–curve environments using Koblitz curves:

Definition 6.8.7. The domain parameters for a 2^h multi–curve environment where $E/\mathbb{F}_{2^n}$ is Koblitz are:

$$\mathcal{V} := \mathcal{F} \cup (\hat{t}, xP)$$

where sending the modified trace $\hat{t}$ is optional, depending on whether it is to be computed using Lemma 5.6.7. Here one has:

- $\mathcal{F}$ describing the field as was given in Definition 6.5.21;

- $a \in \mathcal{F}$ is the curve coefficient from Definition 6.6.21;
- xP is an r–small element from Corollary 6.7.2 which defines the base point $P = (xP, yP)$ with $yP := \min\{\pm yP\}$ computed via the Weierstraß form.

Using the fact that $E : y^2 + xy = x^3 + ax + 1$ has a minimal cofactor $c = 2(2 - a)$ as in Definition 6.6.4, one is now able to define valid parameters for ECC.

Corollary 6.8.8. *Once $\mathcal{F}$ is established, the specific security parameter k is known. Hence for our $\mathcal{V}$ defined above one has*

$$\|\mathcal{V}\| = 10 - a + \lceil s \rceil \,,$$

when using compact tables and

$$\|\mathcal{V}\| = 22 - a + \lceil s \rceil \,,$$

without where one requires 2^s base points P.

By direct calculation for all selected $k \in [162, 570]$ one has that

	$\|\mathcal{V}\|$ for SEC 1	$\|\mathcal{V}\|$	$\|\mathcal{V}_t\|$
k	$\leq 5.381k$	$22 - a + \lceil s \rceil$	$10 - a + \lceil s \rceil$

where $\mathcal{V}_t$ is when using compact tables and $\mathcal{V}$ when not.

6.9 Discussions & Conclusions

If one wants to use a multi–curve model, then many parameter sets may be tied to one system. We assume bandwidth should be minimal here. Using current standards, one can define multi–curve systems but one has no degree of flexibility to optimise a system for how it will be used: The range of security parameters over which is will be deployed or how many systems it is required to specify. In §6.8 we gave a system which overcame these limitations. We defined models for domain parameter representation which represent non–special curves over recommended fields. These systems were bandwidth efficient with respect to how many fields, curves and base–points were required to be specifiable. This gave a system which requires a fraction of the bandwidth used by current propositions. Our model can define domain parameters for prime fields in at–most a fifth of the bandwidth normally expected for a given security parameter k. Better results were achieved when using binary fields.

We gave models for binary and prime fields separately. This is reasonable as one often choose binary fields when deploying in hardware and prime fields when using software. We discuss the advantages of this work for smart–cards and gave methods for defining OEFs which are often used in this setting.

When using binary fields, it is our opinion that one should always use the low–weight polynomials given in §6.5.3. Since there are few of them, and they can be compactly represented in a table, we feel this method would often be optimum.

As our work shows, one can define cryptographic domain parameters far more efficiently than previously published. We do not consider the fixed curve model of Solinas et al in §6.3 to be desirable since it is restrictive, only defined

over prime fields and not as efficient as the non–fixed curve models we have suggested.

Finally, our generation method is well suited for Edwards curves. This is because one has faster arithmetic when the curve coefficients are $c = 1$ with d small. We gave details on how one could implement an analogue of our minimal systems here.

A man's work is nothing but this slow trek to rediscover, through the detours of art, those two or three great and simple images in whose presence his heart first opened.

Albert Camus (1913–1960)

7

Point Compression for Koblitz Curves

7.1 Motivation

Let E be an elliptic curve over $\mathbb{F}_q$. The Elliptic Curve Discrete Logarithm Problem is defined as: Given a point $P \in E(\mathbb{F}_q)$ of large prime order ℓ and a non–trivial point $Q \in \langle P \rangle$, find the value $a \in [1, \ell - 1] \subseteq \mathbb{N}$ such that $Q = [a]P$.

Pollard gave Monte Carlo algorithms to solve the DLP in a generic group $\mathcal{G}$ of prime order ℓ in an expected $\sqrt{\pi\ell/2}$ steps in [62]. For the ECDLP we say:

Definition 7.1.1. An $E/\mathbb{F}_q$ has a *k–bit security level* if the expected running time required by Pollard's methods is greater than 2^k steps.

One has $\#E(\mathbb{F}_q) \approx \ell \approx q$ for cryptographically interesting cases. Hence one expects to achieve a k–bit security parameter for an $E/\mathbb{F}_q$ when $q \approx 2^{2k}$. Under a bit or byte communication model, one has:

Lemma 7.1.2. *For curves over binary fields, $E/\mathbb{F}_{2^n}$, the transmission cost for an affine point $P = (x, y) \in \mathbb{F}_{2^n} \times \mathbb{F}_{2^n}$ requires $n \approx 2k$ bits.*

Proof. One uses the deterministic point compression methods from §4.1.2 and Seroussi's method from §4.1.3. □

Wiener & Zuccherato [87] and Gallant, Lambert & Vanstone [32] showed that one can accelerate Pollard–ρ methods using equivalence classes, defined as the orbit of a point under the action of an efficiently computable group automorphism of $E(\mathbb{F}_q)$. If these equivalence classes are not too large, then one can define a random walk on the set of equivalence classes. Since the set of equivalence classes is smaller than the group itself, one expects the Pollard–ρ algorithm to terminate more quickly. For all elliptic curves $E/\mathbb{F}_q$, they showed that one can use the automorphism $[-]P = -P$ to achieve a $\sqrt{2}$ improvement.

For applications the group law must be efficiently computable. Hence one particular class of curves which are very attractive are Koblitz curves, given in §4.2. However, here one can define equivalence classes of size $2n$,

$$[P] := \{\pm\psi(P) : 0 \leq i < n\},$$

using the Frobenius automorphism $\psi(P)$. This yields a further $\sqrt{n}$ speed–up when using Pollard–ρ; the greatest known for these attacks.

Thus when one uses Koblitz curve systems, one requires that $q \approx 2^{2k+\lg n}$ to achieve a k–bit security parameter. Hence we have a lower security per bit than

when using general curves over the same field. A natural question is to ask is can one achieve a bandwidth for Koblitz systems of $\approx 2k$ bits.

We will answer this question here and present a method to reduce this bandwidth. Subsequently we will show with a low probability of failure, one can compress this bandwidth to the expected number of bytes for a given security parameter k, when using an analogue of the Diffie–Hellman Key Agreement from §2.5.1.

7.1.1 Overhead in Koblitz Systems

We present our communication models and define what we mean by *overhead*:

Definition 7.1.3. The three variants for our communication model is where the receiver expects to get either:

- an m–bit string or,
- an m–byte string or,
- a variable length bit–string of size $\leq m$ where the receiver knows that the string has ended by some end–of–transmission (EOT) symbol, transmission pause or transmission pulse.

Definition 7.1.4. The additional bandwidth required to send a point using a Koblitz system, over a generic one for an equivalent security parameter k is called *overhead*.

Obviously this overhead is dependent on which specific communication model is being utilised.

Lemma 7.1.5. *Let $E/\mathbb{F}_{2^n}$ be a Koblitz curve whose largest prime order ℓ subgroup $\langle P\rangle \subseteq E(\mathbb{F}_{2^n})$ has a k–bit security level. Then the* bit–overhead in the point representation of E *is*

$$r_{bit} \approx [\lg n]. \tag{7.1.1}$$

Equivalently the byte–overhead in the point representation of E *is*

$$r_{byte} \approx \lceil n/8 \rceil - \lceil (n - [\lg n])/8 \rceil \,. \tag{7.1.2}$$

Here $[x]$ is the function returning the nearest integer to $x \in \mathbb{R}$. When it is clear from context we will simply refer to these two quantities as bit and byte redundancies.

Proof. Curves $E/\mathbb{F}_{2^n}$ always have a cofactor $c \geq 2$. Here we assume a general curve has the same cofactor as the Koblitz system; $c \in \{2, 4\}$.

For a fixed security parameter k one expects a Pollard–ρ attack on a general curve to require $\sqrt{\pi\ell/4} > 2^k$ steps. Thus one has $2^{2k} < \ell \leq 2^{2k+1} \Leftrightarrow c \cdot 2^{2k} < \#E(\mathbb{F}_{2^n}) \leq c \cdot 2^{2k+1}$ which implies one requires $(2k + 1 + \lg c)$ bits to represent a point using point compression and Seroussi's method.

The improved Pollard–ρ attack mentioned above implies that one requires $\sqrt{\pi\ell/4n} > 2^k$ for Koblitz systems. Here $2^{2k+\lg n} < \ell \leq 2^{2k+1+\lg n}$ and one needs $(2k + 1 + \lg n + \lg c)$ bits to transmit a point as above.

The difference is clearly $\lg n$ bits and we consider $r_{\text{bit}} \approx [\lg n]$. The equivalent argument for r_{byte} immediately follows. □

Using Lemma 7.1.5 one can compute a table of the bit and byte redundancies when using recommended Koblitz curves from [14]:

Parameters	n	Security level k–bits	r_{bit}	r_{byte}
sect163k1	163	77	7	1
sect233k1	233	112	8	1
sect239k1	239	115	8	1
sect283k1	283	137	8	1
sect409k1	409	199	9	2
sect571k1	571	280	9	1
general curve	n	$\lfloor n/2 \rfloor$	0	0

Table 7.1: Additional Bandwidth required for Koblitz Systems

It is our aim in the sequel to reduce the values of r_{bit} and r_{byte}.

7.2 Reducing Bandwidth: Point Compression

We aim to reduce the values of r_{bit} and r_{byte} from Table 7.1 by working directly with the equivalence classes used in their Pollard–ρ based attacks. In order to work with equivalence classes, one must define a canonical representative of each class. Here we will construct such a representative with a short representation. This will enable us to reduce bandwidth.

For the sequel we define the following: Assume $\mathbb{F}_{2^n}$ is in normal form with a basis $\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{n-1}}\}$ for $\mathbb{F}_{2^n}/\mathbb{F}_2$ and let $E/\mathbb{F}_{2^n}$ be a Koblitz curve whose largest prime order ℓ subgroup is $\langle P \rangle$. A brief summery of the idea we will present here is:

> Extract the abscissa x of a point $P = (x, y)$ and discard its ordinal y. Construct a set of all bit–wise rotations of x. Perform Seroussi's method on this set to save a bit in the representation of the abscissæ. Select a certain representative $\overline{\mathtt{x}}$ from this set which has a short representation. Transmit $\overline{\mathtt{x}}$ instead of P to save bandwidth.

We now present algorithms for compression and decompression using the above idea. Subsequently we detail the theory behind them and the expected theoretical results. Finally we present practical results which support our idea and draw any conclusions required.

7.3 Compression & Decompression Algorithms

We give algorithms here using pseudo–code where $x \in \mathbb{F}_{2^n}$ is represented as a vector $\mathtt{x}$ with respect to some normal basis.

We use the following notation: A binary string $\mathtt{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)$ has a substring $\mathtt{x}[j, i] = (x_j, x_{j-1}, \ldots, x_i)$ and a coefficient $\mathtt{x}[i] = x_i$. Here $\mathtt{x} \ll i$ is the string $\mathtt{x}$ rotated by i places to the left.

Let $t_{\min} \in \mathbb{N}$ be the minimum length of a run of 1's required in the binary representation of an abscissa $\mathtt{x} = \mathrm{x}(P)$ and

$$t_{\min} := \begin{cases} 8 \cdot r_{\text{byte}} - 2 & \text{when using byte–strings;} \\ r_{\text{bit}} - 2 & \text{when using bit–strings;} \\ 1 & \text{when using variable length bit–strings,} \end{cases}$$

for our communication model where we set $t_{\min} = 1$ if $t_{\min} \leq 0$. Then;

Algorithm 7.3.1 (Compressing a Koblitz point $Q \in E(\mathbb{F}_{2^n})[\ell]$).
INPUT:$Q = [a]P$ such that $a \in_R [1, \ell - 1]$;
OUTPUT:A compressed string $\mathtt{x}'$ representing Q or fail.

```
 1.  set x ← x(Q), x̄ ← null, n ← #x, i_0 ← 0 and t ← t_min − 1;
 2.  for (i = 0; i < n; i++){
 3.     if (x[i] == 0 && x[i + 1] == 1){
 4.        set j ← 1;
 5.        for (; j ≤ n − 2; j++){
 6.           if (x[i + j + 1 (mod n)] == 0) break;
 7.        }
 8.        if(j > t){
 9.           set t ← j, i_0 ← i + 1;
10.           set x̄ ← (x ≫ (i − 1))[n − 1, t + 2];
11.        }
12.     }
13.  }
14.  if (x̄ == null) reject Q, halt;
15.  switch (communication model):  //Pad the string x̄
16.     (byte-strings):
17.     (bit-strings):
18.        set x' ← x̄ || 0 || (1)^{t−t_min};
20.     (variable length bit-strings):
21.        set x' ← x̄;
22.  return x';
```

Remarks

- Lines 5 & 6 of the algorithm search for the longest sub–string of the form

  $$0(1)^t 0$$

  where $(1)^t$ is a run of 1's of length t including wrap–arounds.

- Line 8 of the algorithm ensures that we choose the first occurring run of length t, making the selection unique.

- Line 10:

  $$\mathtt{set}\ \bar{\mathtt{x}} \leftarrow (\mathtt{x} \gg (i-1))[n-1, t+2];$$

  is both Seroussi's method for normal representations (see Lemma 4.1.18) and the removal of the run $0(1)^t 0$ of length $(t + 2)$.

- Line 18 pads the truncated string $\overline{\mathtt{x}}$ to the length expected for a generic system for a given security level. That is: we desire a string of length $m = n - (t_{\min} + 2)$ and we have the string $\mathtt{x}' \parallel \mathtt{0}$ of length $(n - t - 3) + 1$ bits. The difference is $(t - t_{\min})$ bits.

Algorithm 7.3.2 (Decompressing a Koblitz point $Q \in E(\mathbb{F}_{2^n})[\ell]$).
INPUT:A compressed string $\mathtt{x}'$ representing Q;
OUTPUT:The abscissa xQ corresponding to a point $Q \in \langle P \rangle$.

```
 1.  set x ← null, j ← 0, s ← 0 and m ← #x';
 2.  for (i = 0; i < m; i++){
 3.     if (x[i] == 0){
 4.        set j ← i;
 5.        break;
 6.     }
 7.  }
 8.  set x ← x'[n − 1, j];
 9.  set x ← x ‖ (1)^{n−m−1+j} ‖ 0;
10.  for (i = 0; i < n − 1; i++){
11.     set s ← (s + x[i]) (mod 2);
12.  }
13.  if (s == a){
14.     set x ← x ‖ 0;
15.  } else {
16.     set x ← x ‖ 1;
17.  }
18.  return x;
```

Remarks

- Lines 2–7 find the original truncated string. Line 9 adds the length t run of 1's and the trailing 0. One now has a $n - 1$ bit string.
- Lines 10–12 compute the trace of the truncated string using Lemma 4.1.18:

$$\mathrm{Tr}(\mathbf{x}) = \sum_{i=0}^{n-1} x_i.$$

  Since $\mathrm{Tr}(a) = \mathrm{Tr}(\mathbf{x})$ here, lines 13–17 reconstruct the original LSB, giving us the original uncompressed abscissa of Q.

7.4 Compressing Koblitz Abscissæ: Theory

In the following subsections we justify that the compression and decompression algorithms given in §7.3 work with a low probability of failure.

7.4.1 Equivalence Classes

Here we present the equivalence classes used for the Pollard–ρ attack on Koblitz systems given in [32, 87]. Let $\overline{R}$ be a representative of the equivalence class $[R]$:

Definition 7.4.1. Let the relation $\sim$ on $E(\mathbb{F}_{2^n})$ be defined by

$$R \sim S \qquad \text{if and only if} \qquad R = \pm\psi^j(S)$$

for some $j \in [0, n-1]$, $R, S \in \langle P \rangle$ with the Frobenius automorphism $\psi(x, y) = (x^2, y^2)$.

Lemma 7.4.2. *For all $Q \in \langle P \rangle$ one has $\psi(Q) \in \langle P \rangle$ if $\ell^2 \nmid \#E(\mathbb{F}_{2^n})$.*

Corollary 7.4.3. *Assume n is prime or $P \notin E(\mathbb{F}_{2^m})$ for any $m \mid n$ where $m < n$. Then definition 7.4.1 trivially forms an equivalence relation on $E(\mathbb{F}_{2^n})$. Moreover from Lemma 7.4.2, $\sim$ partitions $\langle P \rangle$ into equivalence classes*

$$[R] = \{\pm R, \pm\psi(R), \pm\psi^2(R), \ldots, \pm\psi^{n-1}(R)\}$$

of size $2n$ for $R \in \langle P \rangle$ where $R \neq \mathcal{O}$.

Lemma 7.4.4. *Let $Q \in \langle P \rangle$. Then for an $a \in [1, \ell - 1] \subseteq \mathbb{N}$, one has*

$$[a]\big[Q\big] = \big[[a]Q\big].$$

Proof. Point negation $[-] : Q \to -Q$ and the Frobenius are both $E(\mathbb{F}_{2^n})$–group automorphisms: $\pm[a]\psi(Q) = \pm\psi([a]Q) = \psi(\pm[a]Q)$. Thus

$$\begin{aligned}[a]\big[Q\big] &= \{\pm[a]Q, \pm[a]\psi(Q), \ldots, \pm[a]\psi^{n-1}(Q)\} \\ &= \{\pm([a]Q), \pm\psi([a]Q), \ldots, \pm\psi^{n-1}([a]Q)\} = \big[[a]Q\big].\end{aligned}$$

□

7.4.2 Compressing Koblitz Abscissæ up to Rotation

We now present our method to compress bit–strings up to rotation, which represent abscissæ of points $Q \in E(\mathbb{F}_{2^n})[\ell]$ on a Koblitz curve.

First we discuss the bit patterns present in arbitrary binary strings. We then show that one can create an equivalence class of rotated binary strings, using Koblitz abscissæ and the Frobenius automorphism. Subsequently we show that one can use this pattern, to specify a representative of this class. Finally we detail how one could use this to form an analogue to the DHKA.

Definition 7.4.5. Let $\mathtt{x} = \mathtt{x}_{n-1}\mathtt{x}_{n-2}\cdots\mathtt{x}_1\mathtt{x}_0$ represent a binary string with $\mathtt{x}_i \in \{0, 1\}$. We say $\mathtt{x}$ contains a *right padded length t run* if and only if

$$\mathtt{x} = \mathtt{x}_{n-1}\cdots\mathtt{x}_{t+2}\mathtt{011}\cdots\mathtt{110}.$$

That is, the zeroth and the $(t+1)$–th bit are $\mathtt{0}$ with the intermediate t bits being $\mathtt{1}$. Clearly $\#\mathtt{x} \geq 3$ for a run to exist.

Lemma 7.4.6. *Up to shift only three strings of length ≥ 3 do not have a right padded length t run. Namely;*

$$\mathtt{11}\cdots\mathtt{11} = (\mathtt{1})^n, \qquad \mathtt{00}\cdots\mathtt{00} = (\mathtt{0})^n \qquad \textit{and} \qquad (\mathtt{1})^{n-1} \parallel \mathtt{0}.$$

Unless we are in the above three cases, if we find a maximal run of $\mathtt{1}$'s we are guaranteed that these runs are left and right padded by $\mathtt{0}$'s. This result is discussed later with respect to Koblitz abscissæ in Corollary 7.4.15.

Definition 7.4.7. Let $\mathcal{S}_n(\mathtt{x})$ be defined as the set containing all possible bit–rotations of the n–bit binary string $\mathtt{x}$. That is:

$$\mathcal{S}_n(\mathtt{x}) := \{\mathtt{x} \ll i \mid i = 0, 1, 2, \ldots, n-1\}.$$

Corollary 7.4.8. *Let* $\mathrm{x}(P)$ *be the function which extracts the abscissa of a point* $P \in E(\mathbb{F}_{2^n})$*. For the equivalence class* $[R]$ *given in Corollary 7.4.3 one has*

$$\begin{aligned}\mathrm{x}\big([R]\big) &= \mathrm{x}\big(\{\pm R, \pm\psi(R), \pm\psi^2(R), \ldots, \pm\psi^{n-1}(R)\}\big)\\ &= \big\{\mathtt{x} \ll i \mid i = 0, 2, \ldots, n-1\big\} = \mathcal{S}_n(\mathtt{x}).\end{aligned}$$

Proof. The result follows from $-(x, y) = (x, x + y)$ and for a normal basis the Frobenius merely being a bit–shift in the representation of x. □

Definition 7.4.9. Let the *Seroussi function* $s : \{\mathtt{0},\mathtt{1}\}^n \to \{\mathtt{0},\mathtt{1}\}^{n-1}$ take the n–bit abscissa of a point P and return the punctured $(n-1)$–bit string using Seroussi's method. Clearly $s^{-1}(s(\mathtt{x})) = \mathtt{x}$, hence it is well–defined and invertible.

Since all of the strings in $\mathcal{S}_n(\mathtt{x})$ represent abscissæ from $E(\mathbb{F}_{2^n})$, one may use Seroussi's method from §4.1.3 to create a set of n punctured binary strings $S'_n(\mathtt{x})$ of length $n-1$. We now define our representative function f:

Definition 7.4.10. The representative function f is defined by

$$f : \big(\mathcal{S}'_n(\mathtt{x})\big) \to \{\mathtt{0},\mathtt{1}\}^{n-1}$$

where the $\overline{\mathtt{x}} = f\big(\mathcal{S}'_n(\mathtt{x})\big)$ is the $(n-1)$–bit string with the longest right padded length t run in the set $S'_n(\mathtt{x})$. Ties are broken by choosing the smallest $\mathtt{x}_i \in \mathcal{S}'_n(\mathtt{x})$ when considered as an integer.

***Remark* 7.4.11.** f chooses the string with the longest right padded length t run from the punctured set $\mathcal{S}'_n(\mathtt{x})$. If one were to choose this run from the un–punctured set $\mathcal{S}_n(\mathtt{x})$, part of this run may be removed after Seroussi's trick is applied.

Definition 7.4.12. Let the *cutting function* $g : \{\mathtt{0},\mathtt{1}\}^{n-1} \to \{\mathtt{0},\mathtt{1}\}^{n-3-t}$ take a string $\mathtt{x}$ with a right padded length t run and return the string $\mathtt{x}'$ such that

$$\mathtt{x}' \parallel \mathtt{011\cdots110} = \mathtt{x}.$$

Clearly $g^{-1}(g(\mathtt{x})) = \mathtt{x}$, hence it is well–defined and invertible.

Definition 7.4.13. Let $P \in \langle P\rangle$ and $\mathcal{S} = \big(\langle P\rangle/\sim\big)$. *Compression* is the map $\mathcal{C} : \mathcal{S} \to \{\mathtt{0},\mathtt{1}\}^{n-3-t}$ given by

$$g \cdot f \cdot s \cdot \mathrm{x}\big([P]\big) = \mathtt{x}'. \tag{7.4.1}$$

Decompression is the map $\mathcal{D} : \{\mathtt{0},\mathtt{1}\}^{n-3-t} \to \mathcal{S}$ given by

$$\big[\mathrm{x}^{-1} \cdot s^{-1} \cdot g^{-1}(\mathtt{x}')\big] = [P]. \tag{7.4.2}$$

Corollary 7.4.14. *Let* $\mathcal{C}\big([P]\big) = \mathtt{x}'$. *Then one has:*

$$\mathcal{D}\big(\mathcal{C}\big([P]\big)\big) = [P] \qquad \textit{and} \qquad \mathcal{C}\big(\mathcal{D}(\mathtt{x}')\big) = \mathtt{x}'.$$

Proof. From the definitions of $\mathcal{C}$ and $\mathcal{D}$ and $s\big(\mathcal{S}_n(\mathtt{x})\big) = \mathcal{S}'_n(\mathtt{x})$ one has

$$\mathcal{D}\big(\mathcal{C}\big([P]\big)\big) = \mathcal{D}\big(g \cdot f \cdot s \cdot \mathrm{x}\big([P]\big)\big) = \mathcal{D}\big(g \cdot f\big(\mathcal{S}'_n(\mathtt{x})\big)\big) = I.$$

Assume $f\big(\mathcal{S}'_n(\mathtt{x})\big)$ picks a punctured abscissa $\overline{\mathtt{x}}$ corresponding to some $P' \in [P]$;

$$I = \mathcal{D}\big(\mathtt{x}'\big) = \big[\mathrm{x}^{-1} \cdot s^{-1} \cdot g^{-1}(\mathtt{x}')\big] = [\mathrm{x}^{-1} \cdot \mathrm{x}(P')] = [\pm P'] = [P].$$

For the reverse argument, one has: $\mathcal{C}\big(\mathcal{D}(\mathtt{x}')\big) = \mathcal{C}\Big(\big[\mathrm{x}^{-1} \cdot s^{-1} \cdot g^{-1}(\mathtt{x}')\big]\Big)$ and hence $= \mathcal{C}\Big(\big[\mathrm{x}^{-1} \cdot \mathrm{x}(P')\big]\Big) = \mathcal{C}\big([\pm P']\big) = \mathcal{C}\big([P]\big) = \mathtt{x}'$. □

Corollary 7.4.15. *Let* $\mathtt{x}$ *represent the un–truncated* n*–bit binary string representing an abscissa* $x(Q) \in \langle P \rangle$. *Then the forms* $\mathtt{x} = (\mathtt{0})^n$ *and* $(\mathtt{1})^n$ *never occur. Moreover, up to shift only the form* $(\mathtt{1})^{n-1}\mathtt{0}$ *when* $a = 0$ *and* $(\mathtt{1})^{n-2}\mathtt{00}$ *when* $a = 1$ *do not contain any right padded length* t *run.*

Proof. The un–truncated strings represent abscissæ of curves points $Q \in \langle P \rangle$ of large prime order ℓ. Hence $(\mathtt{0})^n$ represents a point in $E(\mathbb{F}_2)$ which never occurs. Similarly the string $(\mathtt{1})^n$ corresponds to the field element $1 \in \mathbb{F}_2$ since this string is fixed under the 2–power Frobenius. Thus this also represents a point in $E(\mathbb{F}_2)$ which never occurs.

The strings that are un–representable are those without a right padded length t run after applying Seroussi's method. From Lemma 7.4.6 these are: $(\mathtt{0})^{n-1}$, $(\mathtt{1})^{n-1}$ and $(\mathtt{1})^{n-2}\mathtt{0}$. Concatenating these with a possible original (punctured) bit $b \in \{\mathtt{0}, \mathtt{1}\}$ gives us 6 strings here: $(\mathtt{0})^{n-1}\mathtt{0}$ and $(\mathtt{1})^{n-1}\mathtt{1}$ which never occur and $(\mathtt{0})^{n-1}\mathtt{1}$ which contains a run of length 1. This leaves $(\mathtt{1})^{n-1}0$, $(\mathtt{1})^{n-2}00$ and finally $(\mathtt{1})^{n-2}01 = (\mathtt{1})^{n-1}0$. These latter 2 strings are un–representable using f. Noting that $\mathrm{Tr}(a) = \sum_{i=0}^{n-1} x_i$ now gives the result. □

7.5 DHKA using Compressed Points

We now will show that one can use the compression and decompression maps $\mathcal{C}$ and $\mathcal{D}$ to construct a Diffie–Hellman Key Agreement protocol:

Initialisation: Let $E(\mathbb{F}_{2^n})$ be a Koblitz curve and $P \in E(\mathbb{F}_{2^n})$ a base–point of maximal prime order ℓ.

Key Exchange:

- Alice picks a random $a \in [1, \ell - 1] \subseteq \mathbb{N}$ and evaluates $Q_A = [a]P$.
- Sets a as her private–key and sends Bob her public–key

$$\mathtt{x}'_A = \mathcal{C}\big([Q_A]\big).$$

- Bob picks a random $b \in [1, \ell - 1] \subseteq \mathbb{N}$ and computes $Q_B = [b]P$.
- Sets b as his private–key and sends Alice his public–key

$$\mathtt{x}'_B = \mathcal{C}\big([Q_B]\big).$$

Key Derivation:

- Alice then computes

$$k_A := \mathcal{C}\big([a]\mathcal{D}(\mathtt{x}'_B)\big)$$

with Bob computing

$$k_B := \mathcal{C}\big([b]\mathcal{D}(\mathtt{x}'_A)\big).$$

Lemma 7.5.1.

$$k_A = k_B.$$

Proof. Alice has $\mathtt{x}'_B = \mathcal{C}\big([Q_B]\big)$ and a. Thus

$$k_A = \mathcal{C}\big([a]\mathcal{D}(\mathtt{x}'_B)\big) = \mathcal{C}\big([a]\mathcal{D}\big(\mathcal{C}\big([Q_B]\big)\big)\big),$$

which from Corollary 7.4.14 and Lemma 7.4.4 one has that this;

$$= \mathcal{C}\big([a]\big[Q_B\big]\big) = \mathcal{C}\big(\big[[a]([b]P)\big]\big) = \mathcal{C}\big(\big[[ab]P\big]\big).$$

Similarly, Bob with $\mathtt{x}'_A = \mathcal{C}\big([Q_A]\big)$ and b has:

$$k_B = \mathcal{C}\big([b]\mathcal{D}(\mathtt{x}'_A)\big) = \mathcal{C}\big([b]\mathcal{D}\big(\mathcal{C}\big([Q_A]\big)\big)\big) = \mathcal{C}\big(\big[[b]Q_A\big]\big) = \mathcal{C}\big(\big[[ba]P\big]\big).$$

□

Remarks

***Remark* 7.5.2.** When using a normal Koblitz system, one specifies the point $Q_i = [i]P \in \langle P \rangle$ in compressed representation $(\mathrm{x}(Q_i), b)$ where b is the bit which indicates which ordinal to use. Here however, one is not required to distinguish a point from its negative since both belong to the same equivalence class.

***Remark* 7.5.3.** In the DHKA protocol above, if Alice (respectively Bob) is unlucky that f cannot find a minimal right padded length t run on her $[Q_a] = \big[[a]G\big]$, she simply chooses another random secret $a \in [2, \ell - 1]$ and tries again.

7.6 Bandwidth Reduction: Theoretical Expectations

By construction, $\mathcal{C}$ selects a well–defined representative truncated punctured abscissa from the points of an equivalence class $[R]$. The advantage here is the function f has n choices for a representative compared with none with the conventional Koblitz curve case: One must just send the point Q_i.

In our construction of the compression function, $\mathcal{C}$ selects such a representative binary string $\mathtt{x}'$ such that some bits are *predetermined* (they contained a right padded length t run). This truncated string $\mathtt{x}'$ was now sent in lieu of P.

A natural question is: How many bits $r = \#\mathrm{x}(P) - \#\mathtt{x}' = n - \#\mathtt{x}'$ can one expect to save under an arbitrary compression function $\mathcal{C}$? We now answer this question:

Theorem 7.6.1. *Let* $\mathtt{x}$ *be a finite binary string of fixed length* n *with* $\mathcal{S}_n(\mathtt{x})$ *defined as above. Assume a function* f *selects an* $\mathtt{x}_i \in \mathcal{S}_n(\mathtt{x})$ *if and only if* r*–bits of* $\mathtt{x}_i$ *are pre–determined. Let the truncated string* $\mathtt{x}'$ *be created by removing these known bits. Then the longest expected truncation by* f *is*

$$r \to \lg n$$

bits as $n \to \infty$.

Proof. Without loss of generality, assume the function f only selects strings $\mathtt{x}_i$ whose leftmost r–bits are fixed in some known form. Hence knowing f and the $(n-r)$–bit truncated string $\mathtt{x}'$ is enough to uniquely reconstruct $\mathtt{x}$.

We are interested in how many n–bit strings are representable by this truncated form: Consider the string

$$\mathtt{x} = \underbrace{\mathtt{x}_{n-1}\mathtt{x}_{n-2}\cdots\mathtt{x}_r}_{\mathtt{x}'}\overbrace{\mathtt{x}_{r-1}\cdots\mathtt{x}_0}^{\mathtt{x}_r}.$$

Trivially for a fixed function f and a fixed $\mathtt{x}_r$, only 2^{n-r} binary strings are representable by $\mathtt{x}$. Thus for a random n–bit string $\mathtt{x}$ the probability that one has a fixed $\mathtt{x}_r$ is

$$\mathbb{P}(\mathtt{x} = \mathtt{x}' \parallel \mathtt{x}_r) = \left(\frac{2^{n-r}}{2^n}\right) = 2^{-r}.$$

We want such a fixed $\mathtt{x}_r$ to occur in at least one string $\mathtt{x}_i$ from the n cyclic–shifts of $\mathtt{x}$, namely from $\mathcal{S}_n(\mathtt{x})$. We have

$$1 \geq \sum_{\mathtt{x}_i \in \mathcal{S}_n(\mathtt{x})} \mathbb{P}(\mathtt{x}_i = \mathtt{x}_i' \parallel \mathtt{x}_r) \approx n2^{-r}.$$

Hence since $r \ll n$, one expects that $r \to \lg n$ as $n \to \infty$ is the best one can achieve if one wants to be able to find a string of the form: $\mathtt{x} = \mathtt{x}' \parallel \mathtt{x}_r$. □

7.7 Bandwidth Reduction: Practical Results

The expected maximal bound for r only holds asymptotically, however one can demonstrate that $(t + 2 = r) \to r_{\text{bit}}$ for even modest (and cryptographically useful) values of n.

Here we present practical results that the probability of a random n–bit string has an encoding by f for integer values of $t \in \mathbb{N}$ across the range

$$1, 2, \ldots, [\lg n] - 2, \ldots, B \qquad \text{for some bound} \qquad B \in \mathbb{N}.$$

That is there exists a right padded length t run somewhere in all possible shifts of $(n-1)$–bit punctured strings of $\mathcal{S}'_n(\mathtt{x})$.

Appropriate code in C# selected random strings of (cryptographically useful) length n, then searched through their orbits and recorded the right padded length t runs as $r = t + 2$. In total $100n$ random strings were tested for each fixed n and the results appear in Table 7.2 below:

	bit–size $(n-1)$						
r	162	232	238	282	408	570	2046
3	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1
6	0.9974	1	1	1	1	1	1
7	0.9480	0.9862	0.9769	0.9915	0.9992	1	1
8	0.7372	0.8612	0.8647	0.9578	0.9655	0.9902	1
9	0.4795	0.6147	0.5887	0.6940	0.8085	0.8965	1
10	0.2778	0.3797	0.3588	0.4454	0.5589	0.6735	0.9847
11	0.1530	0.2164	0.1971	0.2525	0.3324	0.4230	0.8601
12	0.0780	0.1138	0.1050	0.1397	0.1826	0.2465	0.6114

Table 7.2

The black coloured cells indicate the value $r = [\lg n]$ and the grey of $r = \lfloor \lg n \rfloor$ when different. The case of $n = 2^{11} - 1$ has been included purely for theoretical interest to see what happens as n grows beyond what is currently required.

***Remark* 7.7.1.** The C# code here looked for runs of the form $\mathtt{0(1)^t0}$ in an $(n-1)$ bit string since we assume Seroussi's method would always be applied to an n–bit string. This explains the column names in Table 7.2.

Thus any confidence in the table is that of us being able to truncate an n–bit string by $(r+1)$ bits.

7.8 Using a Variable Length Communication Model

In Definition 7.1.3 we defined a variable length communication model. This is where one would send a bit–string of size $\leq n$ where the receiver knows that the string has ended by some end–of–transmission (EOT) symbol, pause or pulse.

This model is particularly interesting when using our abscissæ compression method: Here one would always send the $\mathtt{x}' = \mathcal{C}([P])$ with the longest right padded length t run for optimum bandwidth efficiency. As the results of Table 7.2 show, often one finds right padded length t run longer than required, giving a bandwidth saving.

No known implementation of this model is currently used however, and this result is included for completeness.

7.9 Discussion of Practical Results & Conclusions

The sample size of $100n$ random strings for each fixed n is small from the total $(2^{n-1} - 1)$ strings available. However Table 7.2 does present good empirical evidence that the theoretical bound of $r \to \lg n$ holds as n grows.

For most applications, we are interested in how one can use the confidences presented in Table 7.2 to reduce the byte overhead r_{byte} to zero for the Koblitz curves based systems from §7.5. From Table 7.1 we have:

n	r_{bit}	r_{byte}	r s.t. $r_{\text{byte}} = 0$	$\mathbb{P}_r$
163	7	1	2	1
233	8	1	8	0.8612
239	8	1	6	1
283	8	1	2	1
409	9	2	8	0.9655
571	9	1	2	1

Table 7.3: Observed Probability for Koblitz Point Compression

Here $\mathbb{P}_r$ is the observed probability one can find a length r run from Table 7.2. This implies one can truncate $(r+1)$ bits from an n–bit string as was indicated in Remark 7.7.1.

Corollary 7.9.1. *Using the results of Table 7.3 and Definition 7.4.13 of $\mathcal{C}$, one can truncate with a strong confidence Koblitz abscissæ to the same size as one would expect for non–subfield curves.*

***Remark* 7.9.2.** The results given here can be extended to hyperelliptic curves to create analogous functions $\mathcal{C}$ and $\mathcal{D}$ for enabling point compression although not as efficiently. These are not considered here.

Part II
Disguising

Man cannot discover new oceans, unless he has the courage to lose sight of the shore.

André Gide (1869–1951)

8 Background

8.1 Subgroups and Algebraic Tori

In the sequel we will present cryptographic protocols which work in the subgroup of a multiplicative field. Here we present the important definitions from elementary algebra that we will later require:

Definition 8.1.1. [42] Let $n \geq 1$. The *n–th roots of unity over* $\mathbb{F}_q$ are the roots of the equation

$$x^n - 1 \in \mathbb{F}_q[x].$$

Over $\mathbb{C}$ one has the complete linear factorisation of $x^n - 1$ as

$$x^n - 1 = \prod_{i=0}^{n-1} (x - \zeta_n^i) \tag{8.1.1}$$

for an n–th root $\zeta_n \in \mathbb{C}$.

Theorem 8.1.2. *Let $\mathcal{G}$ be any finite group, $x \in G$, $\mathrm{ord}(x) = n$ and $t \in \mathbb{Z}$. Then $\mathrm{ord}(x^t) = n/\gcd(t, n)$.*

Corollary 8.1.3. *By Theorem 8.1.2, $\mathrm{ord}(\zeta_n^i) = n/\gcd(i, n)$. Thus for $\mathrm{ord}(\zeta_n^i) = d > 0 \Rightarrow d|n$.*

Corollary 8.1.4. [42] *If $\gcd(n, \mathrm{char}(\mathbb{F}_q)) = 1$, then the n–th roots of unity of $\mathbb{F}_q^*$ are $\langle \zeta_n \rangle$. A cyclic group of order n.*

Definition 8.1.5. The *d–th cyclotomic polynomial* $\Phi_d(x)$ is

$$\Phi_d(x) := \prod_{0 \leq i < d = \mathrm{ord}(\zeta_d)} (x - \zeta_d^i). \tag{8.1.2}$$

Lemma 8.1.6. *$\Phi_d(x) \in \mathbb{F}_q[x]$ and $\deg\big(\Phi_d(x)\big) = \phi(d)$.*

Corollary 8.1.7. *From equations* (8.1.1) *&* (8.1.2)*, in $\mathbb{F}_q[x]$;*

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

From Corollary 8.1.7 one has $\Phi_n(x)|(x^n - 1)$. Hence, there exists a subgroup of $\mathbb{F}_{q^n}$ of order $\Phi_n(q)$. Moreover when one has $\Phi_n(q) > n$, Lenstra observed in [43] that for a prime q this subgroup cannot be embedded in any other proper subfield of $\mathbb{F}_{q^n}$. For cryptographically useful groups, this always happens and motivates us to formally define this group:

Definition 8.1.8. The *primitive subgroup* of $\mathbb{F}_{q^n}^*$ is the group of elements

$$G_{n,q} := \{g \in \mathbb{F}_{q^n}^* \mid g^{\Phi_n(q)} = 1\} \subseteq \mathbb{F}_{q^n}^*.$$

We now wish to define *algebraic tori* which will be required in the sequel:

Definition 8.1.9. Let $k = \mathbb{F}_q$ and $L = \mathbb{F}_{q^n}$. Then the *algebraic torus* T_n is the intersection of the kernels of the norm maps $\mathrm{N}_{L/F}$ for all proper subfields $k \subset F \subsetneq L$,

$$T_n(k) := \bigcap_{k \subset F \subsetneq L} \ker\left[\mathrm{N}_{L/F}\right].$$

The following lemma provides some essential properties of algebraic tori

Lemma 8.1.10. [68]

1. $T_n(\mathbb{F}_q) \cong G_{n,q}$,
2. $\#T_n(\mathbb{F}_q) = \Phi_n(q)$,
3. *If $h \in T_n(\mathbb{F}_q)$ is an element of prime order not dividing n, then h does not lie in any proper subfield of $\mathbb{F}_{q^m}/\mathbb{F}_q$.*

We identify algebraic tori with subgroups of $\mathbb{F}_{q^n}^*$ and so write the group operation as multiplication. It should be noted that the sum of two field elements which lie in $G_{n,q}$ does not necessarily lie in $G_{n,q}$, and so we cannot in general add elements of algebraic tori.

We will only be presenting the one dimensional torus T_2 in our research here and not higher dimensional tori. Motivations for this are explained in §9.4.9. Since we only require one model, we can be more explicit defining the torus T_2. First, let us fix some notation: Let $\mathbb{F}_{q^2}$ be some simple extension field defined by $\mathbb{F}_{q^2} := \mathbb{F}_q(\alpha)$ where $\alpha \in \mathbb{F}_{q^2} \setminus \mathbb{F}_q$ is an algebraic element of order 2. Let σ be Frobenius and write $\overline{\alpha} = \sigma(\alpha)$ for the Galois conjugate.

8.2 Rubin & Silverberg's Parameterisation of T_2

Rubin & Silverberg presented a compact parameterisation for tori in [67, 68]. There are two parameterisations depending on whether one is working with binary fields or otherwise. These are analogous, and for simplicity we present only the non–binary case here.

Definition 8.2.1. From Definition 3.1.14 one has: $\mathrm{N}_{\mathbb{F}_{q^2}/\mathbb{F}_q}(x) = x \cdot \sigma(x)$. Then the one dimensional algebraic torus $T_2(\mathbb{F}_q) \subset \mathbb{F}_{q^2}^*$ is directly represented by

$$T_2(\mathbb{F}_q) := \{x \in \mathbb{F}_{q^2} \mid x \cdot \sigma(x) = 1\}. \tag{8.2.1}$$

This is usually called the *affine representation* since one can describe the torus in $\mathbb{A}^2(\mathbb{F}_q)$ using an affine equation; see Definitions 8.2.2 & 8.2.3 later. The equivalent projective representation is:

$$T_2(\mathbb{F}_q) := \left\{ \frac{a + b\alpha}{a + b\overline{\alpha}} \mid a, b \in \mathbb{F}_q, (a, b) \neq (0, 0) \right\} \tag{8.2.2}$$

since there exists a mapping between $\mathbb{P}^1 = \{[a : b]\}$ and T_2 using equation (8.2.4).

Let q be a power of an odd prime. Write $\mathbb{F}_{q^{2m}} = \mathbb{F}_{q^m}(\sqrt{d})$ for some non–square $d \in \mathbb{F}_{q^m}^*$ and let $\sigma \in \mathrm{Aut}(\mathbb{F}_{q^{2m}}/\mathbb{F}_{q^m})$ be the non–trivial isomorphism $\sigma(\sqrt{d}) = -\sqrt{d}$. Under this one can see that for $a, b \in \mathbb{F}_q$;

$$\sigma(a + b\sqrt{d}) = (a + b\sqrt{d})^q = (a - b\sqrt{d}).$$

Hence one can directly represent the affine torus from equation (8.2.1) by:

$$T_2(\mathbb{F}_{q^m}) := \{a + b\sqrt{d} \mid a, b \in \mathbb{F}_q \text{ with } (a^2 - db^2) = 1\} \subset \mathbb{F}_{q^{2m}}^* \tag{8.2.3}$$

as the norm is simply the product of the conjugates. One can now define two maps between T_2 and $\mathbb{F}_{q^m}$ as follows:

Definition 8.2.2. The (decompression) map

$$\psi : \mathbb{A}^1(\mathbb{F}_{q^m}) \setminus \{0\} \to T_2(\mathbb{F}_{q^m})$$

is defined by

$$\psi(a) = \left(\frac{a + \sqrt{d}}{a - \sqrt{d}}\right) = \frac{a^2 + d}{a^2 - d} + \frac{2a}{a^2 - d}\sqrt{d}. \tag{8.2.4}$$

Definition 8.2.3. The (compression) map

$$\varphi : T_2(\mathbb{F}_{q^m}) \setminus \{\pm 1\} \to \mathbb{A}^1(\mathbb{F}_{q^m})$$

acts on an element $\beta = \beta_1 + \beta_2\sqrt{d} \in T_2(\mathbb{F}_{q^m})$ where $\beta_1, \beta_2 \in \mathbb{F}_{q^m}$ such that

$$\varphi(\beta) = \frac{1 + \beta_1}{\beta_2}. \tag{8.2.5}$$

Here $\beta \neq \pm 1$ which implies that $\beta_2 \neq 0$ due to the norm condition. This ensures the map φ is well–defined.

Under Definitions 8.2.2 & 8.2.3, we are able to show that these maps are birational and invertible:

Lemma 8.2.4. *The maps ψ and φ are birational and invertible;*

$$\psi \circ \varphi = \mathrm{id} = \varphi \circ \psi$$

between $T_2(\mathbb{F}_{q^m}) \setminus \{\pm 1\}$ and $\mathbb{F}_{q^m}^*$.

Proof. First, one can immediately identify $\mathbb{A}^1(\mathbb{F}_{q^m}) \setminus \{0\}$ with $\mathbb{F}_{q^m}^*$. Let $a \in \mathbb{F}_{q^m}$, and hence from equation (8.2.4) we have that;

$$\varphi \circ \psi(a) = \varphi\left(\frac{a^2 + d}{a^2 - d} + \frac{2a}{a^2 - d}\sqrt{d}\right) = \frac{1 + \left(\frac{a^2+d}{a^2-d}\right)}{\left(\frac{2a}{a^2-d}\right)} = a.$$

Equivalently from equation (8.2.5) for a $\beta = \beta_1 + \beta_2\sqrt{d} \in T_2(\mathbb{F}_{q^m})$ with $\beta \neq \pm 1$;

$$\psi \circ \varphi(\beta) = \psi\left(\frac{1 + \beta_1}{\beta_2}\right) = \frac{\left(\frac{1+\beta_1}{\beta_2}\right) + \sqrt{d}}{\left(\frac{1+\beta_1}{\beta_2}\right) - \sqrt{d}} = \frac{1 + (\beta_1 + \beta_2\sqrt{d})}{1 + (\beta_1 - \beta_2\sqrt{d})} = \frac{1 + \beta}{1 + \sigma(\beta)} \cdot \left(\frac{\beta}{\beta}\right) = \beta$$

since by definition $\beta \cdot \sigma(\beta) = 1$. Thus ψ and φ define inverse birational maps between $T_2(\mathbb{F}_{q^m}) \setminus \{\pm 1\}$ and $\mathbb{F}_{q^m}^*$. □

Using this we can represent all but two elements of the torus T_2 with just one element of $\mathbb{F}_{q^m}$. The following corollary shows that one can compute the T_2 group law directly on the compressed representatives:

Corollary 8.2.5. *One can compute the torus group law defined in the subgroup $T_2 \subset \mathbb{F}_{q^{2m}}^*$ using solely finite field operations defined over $\mathbb{F}_{q^m}^*$. The map ψ is not even required. Moreover, one can express the torus group law over $\mathbb{F}_{q^m}$ by:*

$$a \star b = \frac{ab + d}{a + b}, \tag{8.2.6}$$

$$a^{\star 2} = \frac{a^2 + d}{2a}, \tag{8.2.7}$$

$$a^{\star -1} = \frac{a - d}{a - 1}, \tag{8.2.8}$$

where $\star$ represents the T_2 group law acting on representatives of torus elements $a \in \mathbb{F}_{q^m}^*$ such that $\psi(a) \in T_2(\mathbb{F}_{q^m})$.

Proof. From the construction of ψ in Definition 8.2.2 with two elements $a, b \in \mathbb{F}_{q^m}$ we have

$$\psi(a) \cdot \psi(b) = \left(\frac{a + \sqrt{d}}{a - \sqrt{d}}\right)\left(\frac{b + \sqrt{d}}{b - \sqrt{d}}\right) = \frac{ab + d + (a + b)\sqrt{d}}{ab + d - (a + b)\sqrt{d}} = \frac{\left(\frac{ab+d}{a+b}\right) + \sqrt{d}}{\left(\frac{ab+d}{a+b}\right) - \sqrt{d}} = \psi\Big(\frac{ab + d}{a + b}\Big)$$

which yields equation (8.2.6). Substituting $b = a$ into (8.2.6) yields (8.2.7).

Let $a^{\star -1}$ be the T_2 group law inverse of a; that is $a \star a^{\star -1} = 1$. Hence from equation (8.2.6) one has

$$1 = \frac{a \cdot a^{\star -1} + d}{a + a^{\star -1}} \qquad \Longleftrightarrow \qquad a^{\star -1}(a - 1) = (a - d)$$

and equation (8.2.8) then immediately follows. □

Thus the T_2 group operation can be simply replaced by the map

$$\star : \mathbb{F}_{q^m} \times \mathbb{F}_{q^m} \to \mathbb{F}_{q^m}^* \qquad \text{where} \qquad \star\,(a, b) \mapsto \frac{ab + d}{a + b}$$

which works with representatives of torus elements in $\mathbb{F}_{q^m}^*$.

8.3 Trace Based Cryptosystems: LUC

LUC, proposed by Lennon & Smith in [79] is a trace based cryptosystem. We will detail the important results for our exposition here, and refer the reader to [64] & [79] for further details.

Rubin & Silverberg's parameterisation from §8.2 uses rational maps into affine space to represent elements of $G_{2,q}$ as elements of T_2. The LUC cryptosystem uses traces over the ground field instead. Consider the primitive subgroup;

$$G_{2,q^m} = \{g \in \mathbb{F}_{q^{2m}}^* | g^{q^m+1} = 1\} \subseteq \mathbb{F}_{q^{2m}}^*$$

and evaluate the $\mathbb{F}_{q^{2m}}/\mathbb{F}_{q^m}$ trace of an element $g \in G_{2,q^m}$,

$$T(g) := \mathrm{Tr}_{\mathbb{F}_{q^{2m}}/\mathbb{F}_{q^m}}(g) = g + \sigma(g) = g + g^{-1} \in \mathbb{F}_{q^m}. \tag{8.3.1}$$

The *symmetric group* $S_2 = \{\mathrm{id}, \sigma\}$ is a group which acts on $G_{2,q}$, and the quotient $G_{2,q}/S_2$ is the set of orbits of the set $G_{2,q}$ under the group S_2. This gives us the map:

$$G_{2,q}/S_2 \to \mathbb{F}_{q^m} \qquad \text{under} \qquad g \mapsto T(g). \tag{8.3.2}$$

In order to do any discrete logarithm based cryptography using these traces, one needs to compute $T(g^a)$ from $\big(T(g), a\big)$ since one can analogously define:

Definition 8.3.1. DLP–LUC: Given $T(g), T(g^a) \in \mathbb{F}_{q^m}$, find $a \in \mathbb{N}$.

To compute such traces one can use the following:

Lemma 8.3.2. *The general form $x^2 - Px + Q = 0$ with not necessarily real roots α and β, satisfies the second order recurrence relation*

$$t_n = Pt_{n-1} - Qt_{n-2} \tag{8.3.3}$$

where $P = \alpha + \beta$ and $Q = \alpha\beta$ are coprime, $t_0 = 2$ and $t_1 = \alpha + \beta$.

Proof. The sequence $\{\alpha^n + \beta^n\}$ has the property that

$$\begin{aligned} P(\alpha^{r-1} + \beta^{r-1}) - Q(\alpha^{r-2} + \beta^{r-2}) &= \alpha^{r-2}(P\alpha - Q) + \beta^{r-2}(P\beta - Q) \\ &= \alpha^{r-2}(\alpha^2) + \beta^{r-2}(\beta^2) \\ &= \alpha^r + \beta^r \end{aligned}$$

since from equation $x^2 - Px + Q = 0$ one has $\alpha^2 = P\alpha - Q$. This satisfies the second order recurrence relation

$$t_n = Pt_{n-1} - Qt_{n-2} \tag{8.3.4}$$

where $P = \alpha + \beta$ and $Q = \alpha\beta$ are coprime, $t_0 = 2$ and $t_1 = \alpha + \beta$. This is in fact the well known Lucas sequence $V_n(P, Q)$, (see [65]). □

Corollary 8.3.3. *Using the second–order Lucas recurrence relation*

$$t_n = t_1 \cdot t_{n-1} - t_{n-2} \pmod{q^m} \tag{8.3.5}$$

with $t_0 = 2, t_1 = T(g^a)$ one can compute $T(g^{an}) = t_n$ from $T(g^a)$ and n.

Proof. Let us first consider computing $T(g^a)$ from a $T(g)$ and $a \in \mathbb{N}$: Let the constant $T(g) := T \in \mathbb{F}_{q^m}$, then from equation (8.3.1) one has $T = g + g^{-1}$. This can be reformulated as;

$$T = x + x^{-1} \Leftrightarrow x^2 - Tx + 1 = 0 \tag{8.3.6}$$

where the roots are the conjugates g and g^{-1} in $\mathbb{F}_{q^{2m}}$.

From Lemma 8.3.2, in equation (8.3.6) one has $P = T$ and $Q = 1$. We are interested in computing $T(g^a)$ which by equation (8.3.1) is simply equivalent to computing $g^a + g^{-a}$ in $\mathbb{F}_{q^m}$. Hence one can define the sequence

$$t_n = t_1 \cdot t_{n-1} - t_{n-2} \pmod{q^m}$$

where $t_0 = 2$ and $t_1 = T$ for which the a–th iteration of computes

$$t_a = \alpha^a + \beta^a = (g)^a + (g^{-1})^a = T(g^a).$$

The result now follows by simply replacing $t_1 = T(g^a)$ and considering the n–th iteration of recurrence relation. □

Hence it is efficient to compute a $T(g^{ab})$ from $(T(g^a), b)$. This enables one to use LUC for DL–based cryptography.

***Remark* 8.3.4.** For a $g, h \in \mathbb{F}_{q^m}$ one usually has $T(gh) \neq T(gh^{-1})$.

Explanation. Let $h \neq 1$. Then from Theorem 3.1.15 one has

$$T(gh) = T(gh^{-1}) \Leftrightarrow T(gh) - T(gh^{-1}) = 0 \Leftrightarrow T(g(h - h^{-1})) = 0 = T(k)$$

for a $k = g(h - h^{-1}) \in \mathbb{F}_{q^m}$. This occurs for approximately q^{m-1} elements k of $\mathbb{F}_{q^m}$ for a fixed g. Hence this holds approximately $1/q$ of the time. □

8.4 Trace Based Cryptosystems: XTR

Lenstra & Verheul in [45, 46] proposed a natural extension of the idea behind LUC called *Efficient Compact Subgroup Trace Representation* which is more commonly known at XTR[1]. XTR uses traces over $\mathbb{F}_{p^2}$ to represent elements of the order $p^2 - p + 1$ subgroup of $\mathbb{F}_{p^6}^*$; the primitive subgroup

$$G_{6,p} := \{g \in \mathbb{F}_{p^6}^* \mid g^{p^2-p+1} = 1\} \subseteq \mathbb{F}_{p^6}^*.$$

Here, $p \equiv 2 \pmod 3$ is a prime such that the cyclotomic polynomial $\Phi(p) = p^2 - p + 1$ has a prime factor $\ell > 6$. Choosing a $g \in \mathbb{F}_{p^6}^*$ of order ℓ now guarantees that this g does not lie in any proper subfield of $\mathbb{F}_{p^6}$ which is necessary for the construction, (see [43]). As in §8.3 where we had $G_{2,q^m} \cong T_2(\mathbb{F}_{q^m})$ we analogously consider the trace of elements of $G_{6,p}$ over $\mathbb{F}_{p^2}$:

Lemma 8.4.1. *The $\mathbb{F}_{p^6}/\mathbb{F}_{p^2}$ trace of an element $h \in G_{6,p} \subset \mathbb{F}_{p^6}^*$ is*

$$T(h) := \mathrm{Tr}_{\mathbb{F}_{p^6}/\mathbb{F}_{p^2}}(h) = h + h^{p-1} + h^{-p} \in \mathbb{F}_{p^2}. \tag{8.4.1}$$

Given $T(h)$, one can compute the conjugates of h as roots in $\mathbb{F}_{p^6}^*$ of the equation

$$X^3 - cX^2 + c^pX - 1 \in \mathbb{F}_{p^2}[X] \tag{8.4.2}$$

with $c = T(h)$.

Proof. Any element $h \in G_{6,p}$ has order dividing $p^2 - p + 1$, thus $p^2 \equiv p - 1 \pmod{p^2 - p + 1}$ and $p^4 = p^2p^2 = (p-1)(p-1) \equiv p^2 \pmod{p^2 - p + 1}$.

The conjugates of $h \in \mathbb{F}_{p^6}^*$ are: $h, h^{p^2} = h^{p-1}$ and $h^{p^4} = h^{-p}$. By definition, the trace of h is the sum of these conjugates and equation (8.4.1) now follows.

Consider the minimal polynomial $m(X)$ of $h \in G_{6,p}$

$$m(X) := \big(X - h\big)\big(X - h^{p-1}\big)\big(X - h^{-p}\big) = X^3 - cX^2 + bX - a$$

where one has $c = h + h^{p-1} + h^{-p} = T(h)$, $a = h^{p+1-p-1} = 1$ and

$$\begin{aligned} b &= hh^{p-1} + hh^{-p} + h^{p-1}h^{-p} \\ &= h^p + h^{1-p} + h^{-1} \\ &= h^p + h^{-p^2} + h^{p^2-p} \\ &= c^p \end{aligned}$$

since $1 - p \equiv -p^2 \pmod{p^2 - p + 1}$. Equation (8.4.2) is now immediate. □

[1] after its acronymic phonetic pronunciation.

Analogous to LUC, one here represents elements of $G_{6,p}$ by their trace. Similarly one loses the distinction between an element h and its conjugates since: $T(h) = T(h^{p-1}) = T(h^{-p})$.

In order to do any discrete logarithm based cryptography using these traces, one needs to compute $T(g^a)$ from $\big(T(g), a\big)$. We now present a method to efficiently compute this after the following definition and lemmata:

Definition 8.4.2. For $c \in \mathbb{F}_{p^2}$ define

$$F(c, X) := X^3 - cX^2 + c^p X - 1 \in \mathbb{F}_{p^2}[X]$$

and define $t_n = \tau(c, n) = \alpha^n + \beta^n + \gamma^n$ for $n \in \mathbb{Z}$, where α, β, γ are the (not necessarily distinct) roots of $F(c, X)$ in $\mathbb{F}_{p^6}$.

Lemma 8.4.3. *From* [46]:

1. $c = t_1$.
2. $t_{-n} = t_{np} = t_n^p$.
3. $t_n \in \mathbb{F}_{p^2}$ *for all* $n \in \mathbb{Z}$.
4. $F(t_n, \alpha^n) = F(t_n, \beta^n) = F(t_n, \gamma^n) = 0$.

Using Lemma 8.4.3 and by immediately identifying the equation from Definition 8.4.2 with that of equation (8.4.2) one has

$$t_n = \alpha^n + \beta^n + \gamma^n = h^n + (h^{p-1})^n + (h^{-p})^n = (h^n)^1 + (h^n)^{p-1} + (h^n)^{-p}.$$

Thus here, $t_n = T(h^n)$. All that remains is to evaluate $\{t_n\}$:

Lemma 8.4.4. *Let the third–order relation for all* $t_i \in \mathbb{F}_{p^2}$ *be defined by*

$$t_{u+v} \equiv t_u \cdot t_v - t_v^p \cdot t_{u-v} + t_{u-2v}$$

for all $u, v \in \mathbb{Z}$ *with the identities*

$$\begin{aligned}
&\text{(I)} && t_{2n} \equiv t_n^2 - 2t_n^p, \\
&\text{(II)} && t_{n+2} \equiv t_1 \cdot t_{n+1} - t_1^p \cdot t_n + t_{n-1}, \\
&\text{(III)} && t_{2n-1} \equiv t_{n-1} \cdot t_n - t_1^p \cdot t_n^p + t_{n+1}^p, \\
&\text{(IV)} && t_{2n+1} \equiv t_{n+1} \cdot t_n - t_1 \cdot t_n^p + t_{n-1}^p
\end{aligned}$$

and $t_0 \equiv 3$, $t_1 \equiv T(h)$ *and* $t_2 \equiv t_1^2 - 2t_1^p$. *Using these and Lemma 8.4.3 one can define an algorithm for computing* t_n *for an arbitrary* $n \in \mathbb{Z}$.

A man who as a physical being is always turned toward the outside, thinking that his happiness lies outside him, finally turns inward and discovers that the source was actually within.

Søren Kierkegaard (1813–1855)

9 Disguising Objects & Black–Boxes

9.1 Introduction to Black–Box Groups

There has been limited research on whether or not it is possible to implement effective **black–box cryptography** by utilising so–called *black–box groups* and a *disguised* representation of their elements [10, 19, 26, 28, 51].

As a motivational example for this research, let us recall the definition of the generic DLP from Definition 2.3.1 which is independent of the specific representation of a group:

Definition. 2.3.1. Let $(G, \oplus)$ be a finite cyclic group of prime order ℓ and let P be a generator of G. The (additive) map

$$\begin{aligned} \varphi : \mathbb{Z}_\ell &\to G \\ t &\mapsto [t]P = \underbrace{P \oplus P \oplus \cdots \oplus P}_{t-\text{times}} \end{aligned}$$

is an isomorphism between $(\mathbb{Z}_\ell, +)$ and $(G, \oplus)$. The problem of computing φ^{-1} is called the *discrete logarithm problem* to the base P. Specifically; given $P, Q \in \langle P \rangle$ determine the $t \in \mathbb{N}$ such that $Q = [t]P$, i.e: Find $t = \log_P Q$.

As was discussed in §2.3, even though up to isomorphism there is only one group $(G, \oplus)$, different representations of $(G, \oplus)$ give different computational abilities to a cryptanalyst, and hence determine the suitability of $(G, \oplus)$ to be used as a cryptographic primitive. For example, one can use sub–exponential *index–calculus* attacks (see §9.2) on a DLP over a finite field $(\mathbb{F}_q^*, \times)$.

It is this family of attacks among others that we wish to remove from a cryptanalyst's arsenal. We will attempt to do this via presenting a representation of a group which allows computation using the group law, but tries to not provide any additional structure which may be exploited by a cryptanalyst.

It is not possible or meaningful to work with truly generic group representations, for reasons described in §9.1.1. However, in the sequel when we use the group $(G, \oplus)$ we assume it is one where its element representation is not publicly defined or useful to a cryptanalyst, as was presented in Definition 2.3.1.

9.1.1 Generic Algorithms

Shoup in [73] generalised earlier work by Nechaev [56] concerning *generic algorithms*; that is an algorithm that does not exploit any property of an elements

representation, other than it can be uniquely represented by a bit string. Shoup clarified and developed these ideas by defining the following:

Definition 9.1.1. [73] Let $\mathbb{Z}_n$ be the additive group of integers modulo n, and let $\mathcal{S} = \{0,1\}^r$ be the set of bit strings of cardinality of at least n (i.e: $2^r \geq n$). An *encoding function* σ is then simply an injective map from $\mathbb{Z}_n$ to $\mathcal{S}$.

Definition 9.1.2. [73] A *generic algorithm* $\mathcal{A}$ for $\mathbb{Z}_n$ on $\mathcal{S}$ is a probabilistic algorithm that behaves as follows: $\mathcal{A}$ takes as input an *encoding list*

$$[\sigma(x_1), \sigma(x_2), \ldots, \sigma(x_k)]$$

where all $x_i \in \mathbb{Z}_n$ and σ is the encoding function of $\mathbb{Z}_n$ to S defined above. As the algorithm executes, it may consult an oracle $\mathcal{O}$ specifying two indices i and j in the encoding list and a sign bit. The oracle then computes an encoding of the $\mathbb{Z}_n$–group operation $\sigma(x_i \pm x_j)$ according to the sign–bit. Then this encoding is appended to the encoding list, which $\mathcal{A}$ always has access to.

Under these definitions, Shoup proved an important theorem which gives a computational lower–bound for the complexity of a generic algorithm for both the DLP and DHP. As an example of this, the DLP in a group G may be rewritten using the notation above as: Given $(\sigma(1), \sigma(t))$, find $t \in \mathbb{Z}_n$. This gave rise to the following:

Theorem 9.1.3. [16] *Let ℓ be the largest prime factor of $n \in \mathbb{N}$. Let $\mathcal{S}$ be the set of binary strings of cardinality at least n. Let $\mathcal{A}$ be a generic algorithm for $\mathbb{Z}_n$ on $\mathcal{S}$ that makes m oracle queries to $\mathcal{O}$, and suppose that the encoding function $\sigma : \mathbb{Z}_n \to \mathcal{S}$ has been chosen at random. The input to $\mathcal{A}$ is $(\sigma(1), \sigma(t))$ where $t \in \mathbb{Z}_n$ is assumed to be random. The output of $\mathcal{A}$ is $v \in \mathbb{Z}_n$. Then the probability that $t = v$ is $O(m^2/\ell)$.*

Clearly Definitions 9.1.1 & 9.1.2 and Theorem 9.1.3 can be extended to cover any group $(G, \oplus)$ when there exists an encoding function σ.

Corollary 9.1.4. *Let $(G, \oplus)$ be a group of order $n > \ell$ whose largest prime subgroup has order $\ell \mid n$. Then any algorithm $\mathcal{A}_{\mathcal{L}}$ for computing the discrete logarithm in G requires at least $\Omega(\ell^{1/2})$ group operations.*

Proof. In order for one to be assured a non–trivial probability of success of finding the discrete logarithm, one would run $\mathcal{A}$ at least $m = O(\ell^{1/2})$ times. □

9.1.2 Black–Box Groups

Corollary 9.1.4 motivates one to find a representation of groups for which no algorithms perform better than generic algorithms. If one could give such a 'generic' group description for DLP computation, then generic attacks are essentially the best a cryptanalyst can mount, and these are all fully exponential.

Using §9.1.1 we are able to construct formal definitions of a black–box group, black–box–trapdoor group (BBG) and other definitions that we use extensively in the sequel:

Definition 9.1.5. Let $(G, \oplus)$ be a group of order n and $\mathcal{S} = \{0,1\}^r$ the set of bit–strings of cardinality at least $n \geq 2^r$. Let ψ be an (usually specific) injective encoding function

$$\psi : G \to \mathcal{S}.$$

Let $\mathcal{A}$ denote an algorithm that computes the group operation of G on representations in $\mathcal{S}$. That is; for a given $x_1, x_2 \in \mathrm{Im}(\psi) \subseteq S$, $\mathcal{A}$ computes

$$\psi\Big(\psi^{-1}(x_1) \oplus \psi^{-1}(x_2)\Big).$$

Then the resulting algebraic structure $(\mathcal{S}, \mathcal{A})$ is denoted a *black–box group*.

Hence a BBG is one with an algorithm for computing the group law which makes limited use of the group description. This is in contrast to a generic algorithm which does not attempt to use any additional structure of the group, even if the specific representation makes such structure readily available. Thus we now try and find BBG representations that are efficiently implementable on a computer.

Motivation 9.1.6 (Natural Representation)**.** It is difficult to define what a *natural representation* is rigourously, so we describe it using examples. A group $(G, \oplus)$ is said to be have *natural representation* if it is given using a "standard" mathematical representation, such as: $\mathbb{F}_q[x]/\langle f(x)\rangle$ for a finite field, the Weierstraß equation for an elliptic curve $E/\mathbb{F}_q$ et cetera.

Thus a user who possesses the natural representation enjoys the 'normal' computational ability one would expect.

Trivially, a particular natural representation may not be unique. E.g. one can usually find a polynomial $g \neq f$ such that $\mathbb{F}_p[x]/\langle f(x)\rangle \cong \mathbb{F}_p[y]/\langle g(y)\rangle$.

Motivation 9.1.7 (Disguised Representation)**.** We would like to take a natural representation of a group and *disguise* it to get a new representation. Then some structures/features which were evident when using the natural representation are now lost. We will call this new representation a *disguised representation* of the group.

Essentially a disguised representation along with a disguised rule for computing the original group law will be the basis of our construction of a BBG. It is the purpose of our research here to construct examples of such disguised representations.

Definition 9.1.8 (Disguising)**.** *Disguising* is any method which allows one to construct a candidate BBG from a group's natural representation.

Using these definitions we can construct an implementable model for a black–box group. Any finite abelian algebraic group G can have both its elements and group law described using polynomials. This description, which is efficiently implementable on a computer, forms a natural representation of G.

We now construct a specific encoding function $\psi : G \to \mathcal{S}$. We need an injection which maps polynomials to polynomials, and one which ensures a polynomial–time algorithm exists for computing the group law on $\mathcal{S}$ (thus a random map ψ would not suffice). Under this one can compute and publish the set of polynomials which describe the G–group operation on $\mathcal{S}$.

This gives rise to the following important definition:

Definition 9.1.9 (Black–Box Trapdoor Group)**.** Let $X = (G, \oplus)$ be a naturally represented abelian algebraic group of order n. Let $\mathcal{S}$ be some set with cardinality greater than $\#G$, not necessarily $\{0,1\}^r$. Let ψ be an injective encoding function,

$$\psi : G \to \mathcal{S},$$

called the *disguising function* on X. Both G and the trapdoor ψ form the system private–key.

A public disguised representation of X is then computed as follows: Under ψ, for all $g_i \in G$ there exists a $\hat{g_i} \in \mathcal{S}$ where $\psi(g_i) = \hat{g_i}$. Using two algebraic elements $g_1, g_2 \in G$ an equivalent description of the G–group law $\oplus$ is calculated for $\mathcal{S}$ by computing

$$\star(\hat{g_1}, \hat{g_2}) = \star\big(\psi^{-1}(g_1), \psi^{-1}(g_2)\big) = \oplus(g_1, g_2).$$

The resulting algebraic structure $Y = (\mathcal{S}, \star)$ is denoted a *black–box trapdoor group*.

Henceforth, when we refer to black–box groups (BBGs) we will always mean the trapdoor variant given in Definition 9.1.9 above.

***Remark* 9.1.10.** Note that as ψ is only injective, not all $x_i \in \mathcal{S}$ may have a valid relation $x_i = \psi^{-1}(g_i)$ for a $g_i \in G$. That is, a random x_i may not represent a disguised element at all.

***Remark* 9.1.11.** In practice, it will often be required that the inverse $\psi^{-1} : \mathcal{S} \dashrightarrow G$ must also be efficiently computable.

In the remainder of this chapter, we will use the notation $\hat{g}$ to denote the disguised representative of g under ψ similar to that used in Definition 9.1.9.

9.1.3 Security

It is the aim of researchers in this field to improve the operating security (the lower bound of computational complexity for the best known attack) of a cryptographic primitive or to gain new functionality by disguising its representation as described above.

It is hoped here that the disguised representation Y, given in Definition 9.1.9 gives the cryptanalyst a very restricted arithmetic functionality unlike what he would enjoy in a naturally represented X. Under this, working with Y would essentially be equivalent to working with a generic group: From Corollary 9.1.4, one would expect the best attack on a DLP using Y to be exponential *regardless* of whether sub–exponential attacks existed for X. Thus, one would have the 'optimum' security for a given primitive. This is the key motivation for considering BBGs.

In order for us to have this ideal situation, one requires that un–disguising Y needs an infeasible amount of work by the cryptanalyst. Before defining this infeasibility, we first need to define what would make a valid attack.

***Remark* 9.1.12.** The group X is disguised in order to prevent the cryptanalyst utilising a natural representation for an attack, such as index calculus. Hence, a successful attack is one which gives an isomorphism from the BBG to a group in natural representation. This does not have to be exactly the same representation as used by the entity who set up the system, but it should be a representation of the same type (i.e., polynomial representation for a finite field, Weierstraß equation for an elliptic curve et cetera).

Therefore, the goal of an attacker is to compute an isomorphism from Y to *any* natural representation X'. This is called 'looking inside the box' and leads us to the following formalisation:

Definition 9.1.13. A *valid attack* for the disguised system

$$X = (G, \oplus) \underset{\psi}{\longrightarrow} (\mathcal{S}, \star) = Y$$

uncovers *any* natural representation for $X' = (H, \otimes)$, where one has the equivalent level of arithmetical functionality as if given X itself.

This now enables us to define a *securely disguised* BBG representation:

Definition 9.1.14. Let X be a natural representation of a group and Y its disguised form under

$$X = (G, \oplus) \underset{\psi}{\longrightarrow} (\mathcal{S}, \star) = Y.$$

Y is said to be *securely disguised* if given Y, it is hard (viz. §2.2.1) to recover any natural representation X' of X without knowledge of the private–key ψ.

***Example* 9.1.15.** Let the natural representation

$$X = \left(\mathbb{F}_q^* \cong (\mathbb{F}_p[x]/\langle f(x) \rangle)^*, \times \right)$$

be disguised under ψ to give a $Y = (\mathcal{S}, \otimes)$. Then without ψ, it should be hard to recover any isomorphic representation $(\mathbb{F}_p[x]/\langle g(x) \rangle)^*$ to the field $\mathbb{F}_q$.

9.1.4 Computing Maps Between Representations

If one is able to find a natural representation $X' = (\mathbb{F}_q[x]/\langle g(x) \rangle, \times)$ of the disguised representation $Y = (\mathcal{S}, \star)$, one then needs to map the original disguised problem to this natural representation. This is done is a similar way to computing isomorphisms between finite fields given in §3.3: One simply decomposes the elements of the public–basis of Y in terms of the new basis of X'.

9.1.5 Previous Research

The original proposal of disguising elliptic curves was given by Frey [26]. Frey suggested taking an elliptic curve E defined over some fixed extension field, and then by using Weil decent describe this curve as an abelian variety. This set of non–linear multivariate polynomials defined over the ground field now described the original elliptic curve's group operation. This algebraic group was then disguised by applying an invertible transformation, which was essentially just a linear change of variable. From this disguised model it is not necessarily clear how to compute the (elliptic curve) group order or recover a natural representation; a Weierstraß equation for E. As this specific example is not the focus of our research here, further details can be found in [26].

Dent & Galbraith in [19] built on this idea to create a black–box group by hiding pairings for (supersingular) elliptic curve groups. The authors noted however that the parameters required to implement a secure system would be of an impracticable size, due to work by Faugère & Perret at Eurocrypt 2006. Here Faugère used his own F_5 algorithm to attack the *Polynomial Isomorphism Problem with One Secret*, (IP1S) in [24, 59]. These Gröbner based attacks are not known to affect the general problem of a disguised elliptic group, but do affect the hidden pairings problem which was presented in [19].

9.1.6 Our Contribution

We attempt to further the research for suitable methods for implementing BBGs. Section 9.1.5 gives some justification for the study of different algebraic structures to assess their suitability as BBGs: In §9.4.1 we present Galbraith's parameterisation of T_2. This describes the group law as a set of polynomials over a finite field which gives a poorly disguised group. We further this analysis by using some of the rich algebraic structure available for the torus in §9.4.6. We do this by reformulating the disguised group law on T_2 in terms of solving a system of linear equations by using Rubin & Silverberg's parameterisation of the torus from §8.2. We show that such a disguise is still easily uncovered and then advance to discuss how these results would affect disguising T_6.

Finally we show how these methods extend to the trace based methods of LUC given in §8.3 and XTR given in §8.4. We show how these fail to construct suitable BBGs and draw conclusions on BBG–cryptography in general.

9.2 Disguising Finite Fields

Before we describe the disguising of more complex algebraic objects, we start with the simplest case: Disguising a finite field. When a DLP is implemented over a finite field, it is well known that sub–exponential attacks of complexity

$$O\big[L_q(1/3,c)\big] := O\Big(e^{(c+o(1))(\ln q)^{\frac{1}{3}}(\ln\ln q)^{\frac{2}{3}}}\Big)$$

for some constant $c \in \mathbb{R}_{\geq 0}$ exist. These index–calculus methods use the Number Field Sieve [18] where Coppersmith in [17] showed that theoretically one has a $c = \frac{1}{3}(92+26\sqrt{13})^{1/3} \approx 1.9$. Index–calculus methods exploit element representations, thus the hope is a disguised representation will prevent such an attack, making a DLP more secure. Much of what we now discuss follows [28].

Let $K = \mathbb{F}_{q^m}$ be some finite field where q is a prime power. Let $\mathbb{A}^m(\mathbb{F}_q) \cong \mathbb{F}_q^m$ denote m–dimensional affine space over $\mathbb{F}_q$ where an element $a \in \mathbb{F}_{q^m}$ is represented by the m–tuple $\underline{a} = (a_0, a_1, \ldots, a_{m-1})^{\mathrm{T}}$ with respect to some ordered basis $\mathfrak{B} = \{\xi_0, \ldots, \xi_{m-1}\}$ of $\mathbb{F}_q^m$ (for example; a polynomial–basis). Here, we denote that a field element a is represented in m–tuple form by underlining it; $\underline{a}$. Let $\mathcal{B}$ represent the m–dimensional basis–vector of $\mathfrak{B}$, then using this we write $a \sim \underline{a}$ when $a = \underline{a} \cdot \mathcal{B}$ where $\cdot$ denotes the usual matrix dot–product.

9.2.1 Construction

It is easy to explicitly describe the operation of multiplication in the field in terms of m–tuples. One way is to construct m quadratic homogenous polynomials

$$M_i(x_0, x_1, \ldots x_{m-1}, y_0, \ldots, y_{m-1})$$

over $\mathbb{F}_q$ for $i = 0, 1, \ldots, m-1$ such that if $a \sim \underline{a} = (a_0, a_1, \ldots, a_{m-1})^{\mathrm{T}}$ and $b \sim \underline{b} = (b_0, b_1, \ldots, b_{m-1})^{\mathrm{T}}$ for $a, b \in \mathbb{F}_q$, then $\underline{c} \sim c = ab$ is given by

$$c_i = M_i(\underline{a}, \underline{b}) \in \mathbb{F}_q[\underline{x}, \underline{y}]. \tag{9.2.1}$$

Thus, one can fully describe the group law on $\mathbb{F}_q^m$ by the vector M of polynomials

$$\underline{c} = \mathrm{M}(\underline{a}, \underline{b}) = \big(M_0(\underline{a}, \underline{b}), \ldots, M_{m-1}(\underline{a}, \underline{b})\big) \tag{9.2.2}$$

whose elements lie over $\mathbb{F}_q$.

It is this natural representation $X = (\mathbb{F}_q^m, \mathrm{M})$ of $\mathbb{F}_{q^m}$ that we wish to disguise. To do this, we follow the idea of Frey in [26] and simply apply a change of basis transformation onto the vector space $\mathbb{F}_q^m$.

Let the disguising function $\psi : X \to Y$ be defined by generating a random invertible linear transformation $\mathrm{U} \in \mathrm{GL}_m(\mathbb{F}_q)$ such that $\psi(\underline{a}) = \mathrm{U}\underline{a}$ $(= \underline{\hat{a}})$. Here $\mathrm{GL}_m(K)$ is the group of $m \times m$ invertible matrices in K. Under this one has

$$\mathrm{U} : X = \left(\mathbb{F}_q^m, \mathrm{M}\right) \to \left(\mathbb{A}^m(\mathbb{F}_q), \mathrm{N}\right) = Y$$

where Y denotes the disguised representation. The disguised group law N is then a system of m–polynomials which are computed from the set of equations in (9.2.2) as follows:

$$\mathrm{N}(\underline{\hat{a}}, \underline{\hat{b}}) = \mathrm{UM}(\mathrm{U}^{-1}\underline{\hat{a}}, \mathrm{U}^{-1}\underline{\hat{b}}) \tag{9.2.3}$$

where juxtaposition denotes normal matrix multiplication and $\underline{\hat{c}} = \mathrm{N}(\underline{\hat{a}}, \underline{\hat{b}})$.

Corollary 9.2.1. *By construction, the algebraic structure $Y = (\mathbb{A}^m(\mathbb{F}_q), \mathrm{N})$ which describes the finite field multiplication group law on generic elements is a BBG under Definition 9.1.9.*

To compute with $Y = (\mathbb{A}^m(\mathbb{F}_q), \mathrm{N})$, a public/BBG user would take two m–tuples $\underline{\hat{x}}, \underline{\hat{y}}$ over $\mathbb{F}_q$ and use them to evaluate $\mathrm{N}(\underline{\hat{x}}, \underline{\hat{y}})$. Thus a public user is able to compute the disguised group law of X (when $\underline{\hat{x}}, \underline{\hat{y}}$ represent valid disguised elements) without directly using the natural representation; specifically the basis $\mathcal{B}$ of $\mathbb{F}_q^m \in X$. Such computations are efficient since elements of N are simply quadratic polynomials over $\mathbb{F}_q$ (as U is linear) and the addition of elements in Y is trivial to compute.

9.2.2 Cryptanalysis of Disguised Finite Fields

The disguised group $Y = (\mathbb{A}^m(\mathbb{F}_q), \mathrm{N})$ above can be trivially un–disguised, uncovering a natural representation for the finite field. As noted in Definition 9.1.13, a successful attack on a BBG does not need to recover the *exact* natural representation of the original group. Thus here one only needs to construct an isomorphic description of the original field $\mathbb{F}_{q^m} \in X$.

Before we present this attack, let us define some useful lemmata: Assume a random m–tuple $\underline{\hat{w_1}}$ represents some unknown and undisguised element $\underline{w_1} \in X$:

Lemma 9.2.2. *Using $\underline{w_1}$ and $\mathrm{N} \in Y$ one can construct the set*

$$\Delta = \{\underline{\hat{w_1}}, \underline{\hat{w_2}}, \ldots, \underline{\hat{w_{m+1}}}\} = \{\mathrm{U}\underline{w_1}, \mathrm{U}\underline{w_1^2}, \mathrm{U}\underline{w_1^3}, \ldots, \mathrm{U}\underline{w_1^{m+1}}\}.$$

Proof. Let $\mathrm{U}^{-1}\underline{\hat{w_i}} = \underline{w_1^i}$ where $\underline{w_1^i}$ signifies the vector representation of $(\underline{w_1} \cdot \mathcal{B})^i \in \mathbb{F}_{q^m}$. Then using the description of N one can construct

$$\{\underline{\hat{w_1}}, \underline{\hat{w_2}}, \ldots, \underline{\hat{w_{m+1}}}\} = \{\mathrm{U}\underline{w_1}, \mathrm{U}\underline{w_1^2}, \mathrm{U}\underline{w_1^3}, \ldots, \mathrm{U}\underline{w_1^{m+1}}\}$$

by iteratively evaluating $\underline{\hat{w_i}} = \mathrm{N}(\underline{\hat{w_{i-1}}}, \underline{\hat{w_1}})$, giving us a set Δ of $m+1$ vectors in a dimension m vector space. □

Lemma 9.2.3. *A linear dependence on the elements of $\Delta \subseteq Y$ invokes a linear dependence on any natural representation of X.*

Proof. Given $\sum_{j=1}^{m+1} a_j \underline{\hat{w_j}} = \underline{0}$ we have

$$\underline{0} = \mathrm{U}^{-1}\underline{0} = \mathrm{U}^{-1} \sum_{j=1}^{m+1} a_j \underline{\hat{w_j}} = \sum_{j=1}^{m+1} a_j \mathrm{U}^{-1} \underline{\hat{w_j}} = \sum_{j=1}^{m+1} a_j \underline{w_1^j}.$$

Since by construction all elements of Δ represent valid disguised elements of the natural representation, we are done. □

Lemma 9.2.4. *Using* $\mathrm{N} \in Y$ *and Lemma 9.2.3, one can construct the degree* m *polynomial*

$$g(x) = \sum_{j=1}^{m+1} a_j x^{j-1} \tag{9.2.4}$$

over $\mathbb{F}_q[x]$ *which has the undisguised* $w_1 \sim \underline{w_1}$ *as a root.*

Proof. Δ from Lemma 9.2.2 gives us $m+1$ vectors in a dimension m vector space. Thus the elements of Δ must form a linear dependence. By linear algebra one can compute the coefficients $a_j \in \mathbb{F}_q$ of this dependence

$$\sum_{j=1}^{m+1} a_j \underline{\hat{w_j}} = \underline{0}$$

and using Lemma 9.2.3 one now has

$$\underline{0} = \sum_{j=1}^{m+1} a_j \underline{\hat{w_j}} = \sum_{j=1}^{m+1} a_j \underline{w_j} = \sum_{j=1}^{m+1} a_j \underline{w_1^j}.$$

Hence with respect to any basis $\mathcal{B}$ of $\mathbb{F}_q^m$ and since $(\sum \underline{x}) \cdot \mathcal{B} = \sum (\underline{x} \cdot \mathcal{B})$ we have

$$\sum_{j=1}^{m+1} a_j (\underline{w_1^j} \cdot \mathcal{B}) = \sum_{j=1}^{m+1} a_j w_1^j = 0 \in \mathbb{F}_{q^m}.$$

After removing a trivial factor of w_1, one can then define

$$g(x) = \sum_{j=1}^{m+1} a_j x^{j-1} \in \mathbb{F}_q[x]$$

which by construction has the root $w_1 \sim \underline{w_1}$. □

Lemma 9.2.5. *Let* α *be an arbitrary element of* $\mathbb{F}_{q^m}$*, then:*

1. *The minimal polynomial of* α *over* $\mathbb{F}_q$ *exists and is unique; moreover it is irreducible over* $\mathbb{F}_q$*.*
2. *Let* $h(x)$ *be the minimal polynomial for* α *over* $\mathbb{F}_q$*. If* $f(x) \in \mathbb{F}_q[x]$ *and* $f(\alpha) = 0$ *then* $h(x) \mid f(x)$*.*

Proof. See [85]. □

Lemma 9.2.6. w_1 *does not lie in any proper subfield of* $\mathbb{F}_{q^m}$ *if and only if* $g(x)$ *is irreducible and of degree* m*.*

Proof. ($\Rightarrow$) Assume $w_1 \in \mathbb{F}_{q^m}$ does not lie in any proper subfield $\mathbb{F}_q \subseteq F \subset \mathbb{F}_{q^m}$ and let $g(x)$ be the one constructed in Lemma 9.2.4. It is trivial to make $g(x)$ monic, so one has a degree m monic polynomial with w_1 as a root.

By (2) of Lemma 9.2.5, there exists a minimal polynomial $h(x) \in \mathbb{F}_q[x]$ for w_1 such that $h(x) \mid g(x)$ since $g(w_1) = 0$ by construction. Due to the uniqueness of minimal polynomials ((1) of Lemma 9.2.5), either we have $h(x) = g(x)$ or $h(x)$ must be some non–trivial factor of $g(x)$.

Assume the latter is true: That is one has $n = \deg(h) < \deg(g)$ and using this we can construct the field $F_1 := \mathbb{F}_q[x]/\langle h(x)\rangle \cong \mathbb{F}_{q^n}$ which has $w_1 \in F_1$. This contradicts w_1 not lying in any proper subfield of $\mathbb{F}_{q^m}$. Hence $g(x) = h(x)$, and so $g(x)$ is trivially irreducible and of degree m as required.

($\Leftarrow$) If $g(x)$ is irreducible, by the removal of its leading coefficient one is able to construct an irreducible monic degree m polynomial with w_1 as a root. It follows immediately from Lemma 9.2.5 that $g(x)$ must be the unique minimal polynomial for w_1 and hence cannot lie in any subfield of order less than q^m. $\square$

Using these lemmata we are now in a position to define an attack on these 'disguised' finite fields:

9.2.3 Attack Algorithm

An attack to uncover an equivalent natural representation of X from Y is:

Algorithm 9.2.1 (Attack of a Disguised Finite Field $\mathbb{F}_q$)**.**

1. Acquire the disguised system $Y = (\mathbb{A}^m(\mathbb{F}_q), \mathrm{N})$ and generate a random m–tuple over $\mathbb{F}_q$, $\underline{\hat{w_1}}$.

2. Using the disguised group operation $\mathrm{N} \in Y$ compute the set

$$\Delta = \{\underline{\hat{w_1}}, \underline{\hat{w_2}}, \ldots, \underline{\hat{w_{m+1}}}\} = \{\mathrm{U}\underline{w_1}, \mathrm{U}\underline{w_1^2}, \mathrm{U}\underline{w_1^3}, \ldots, \mathrm{U}\underline{w_1^{m+1}}\}$$

   by iteratively evaluating $\underline{\hat{w_i}} = \mathrm{N}(\underline{\hat{w_{i-1}}}, \underline{\hat{w_1}})$. Using Lemma 9.2.4 one can now construct the degree m polynomial

$$g(x) = \sum_{j=1}^{m+1} a_j x^{j-1}$$

   over $\mathbb{F}_q$ from equation (9.2.4).

3. Test if $g(x)$ is reducible. If `TRUE` then Lemma 9.2.6 implies that our w_1 lies in some proper subfield of the unknown $\mathbb{F}_{q^m}$ and hence does not generate a polynomial–basis for $\mathbb{F}_q^m$. Here we reject $\underline{\hat{w_1}}$ and return to `STEP 1`.

4. Else, Lemma 9.2.6 implies we now have the minimal polynomial (by trivially removing g's leading coefficient) of degree m over $\mathbb{F}_q$ which allows us to construct the field extension

$$\mathbb{F}_{q^m} \cong \mathbb{F}_q[x]/\langle g(x)\rangle$$

   which is an isomorphic to the one contained in the undisguised X.

5. One now has a natural representation $X' = (\mathbb{F}_q[x]/\langle g(x)\rangle, \times)$ and so enjoys the same arithmetical ability that the holder of X does. All one needs to do is to map the cryptographic problem from $Y \to X'$. To do this one simply decomposes the elements of the public–basis in terms of the new polynomial–basis which can be done easily using §9.1.4.

Hence the public system Y is now undisguised and has no additional security than if the problem was directly given over X. This algorithm constructs a valid attack as defined in Definition 9.1.13.

***Remark* 9.2.7.** The probability of success of the algorithm is 1, because it does not stop until it finds an irreducible polynomial in `Step 3`. However, the algorithm may run forever if it keeps making poor choices for $\underline{\hat{w}_1}$.

Hence here we can only consider the expected running–time for this algorithm: $\mathrm{N}(\underline{a}, \underline{b})$ has m^2 elements each of which are evaluated by a field multiplication and summation. Hence `Step 2` requires us make m evaluations of N, costing $O(m^3 \lg^2 q)$ operations in $\mathbb{F}_q$. After trivially constructing $g(x)$ from Δ, one must test it for irreducibility. Using the *Frobenius Iterated Distinct–Degree factorisation algorithm* from ([83], p390), one expects this to take $\tilde{O}(m^2 + m \lg q)$ $\mathbb{F}_q$–operations. At a maximum one expects m different subfields, hence one expects to run `Steps 1–3` at most m/q times since any element has index $1/q$ in the whole field. Overall the expected run–time of the algorithm is $\tilde{O}(m^3 \lg^2 q)$ operations in $\mathbb{F}_q$.

***Remark* 9.2.8.** If this were to be implemented, for security reasons the extension field would not contain too many proper subfields. Thus one would expect the algorithm to usually run for the expected time given in Remark 9.2.7.

***Remark* 9.2.9.** The field considered here is a black–box group but not a true black–box *field* as the linearity of U does not disguise field addition. Hence, this attack does not contradict the results of Boneh & Lipton in [10].

***Remark* 9.2.10.** The disguised system Y has two stationary points which relate to the undisguised relations $0 \cdot 0 = 0$ and $1 \cdot 1 = 1$. Hence using $\mathrm{N} \in Y$ one can mount an alternative attack by finding and storing fixed points generated by solving the equations

$$N_i(\underline{\hat{x}}, \underline{\hat{x}}) = \underline{\hat{x}}$$

using Gröbner bases methods with the $N_i \in \mathrm{N}$. Since the attack presented here is far more trivial we do not go into detail.

One of the principal reasons that this attack is possible, is the attacker can use the public group operation matrices in any way he chooses, not just as a disguised multiplication oracle. This theme will appear again later in the sequel.

9.3 Disguising Using Affine Transformations U

Definition 9.3.1. An *affine transformation* U between two vector spaces is a linear transformation followed by a translation, namely:

$$\underline{x} \mapsto \mathrm{M}\underline{x} + \underline{t}.$$

Trivially an affine transformation is invertible if and only if M is. Frey in [26] originally proposed disguising elliptic curves by applying an affine invertible transformation $\mathrm{U}(\underline{x}) = \mathrm{M}\underline{x} + \underline{t}$ for a secret change-of-basis matrix M and vector $\underline{t}$ on the vector space $\mathbb{F}_q^m$.

However, Perret in [59] observed that $\mathrm{M}\underline{x} + \underline{t}$ was equivalent to $\mathrm{M}'(\underline{x}')$ where $\underline{x}' = (\underline{x}, 1)$ and

$$\mathrm{M}' = \begin{pmatrix} \mathrm{M} & \underline{t} \\ \underline{0} & \underline{1} \end{pmatrix}.$$

Hence using affine transformations does not increase security at the cost of increasing complexity, especially in an exposition. For these reasons, affine maps are not considered in the sequel.

Perret's method does not work for non–linear transformations of higher degrees, and such maps could make the disguised form more complex and hence difficult to attack. However, when using such maps the group law would be inefficient to compute with and to specify, due to the (expected) high–degree polynomial equations which would be required describe it. When one wished to invert a disguised representation, one must also deal with non–uniqueness of undisguised elements. Neither of the latter would make such a system desirable from an implementation aspect. These maps are similarly disregarded from our work in the sequel.

9.4 Disguising Tori

It was shown in §9.2.2 that disguised finite fields do not make suitable algebraic structures for disguised cryptography. In this section, we extend this research by investigating whether algebraic tori make more suitable candidates for the construction of a BBG. We first present the work by Galbraith from [28]. In this he gave a method to disguise the one dimensional torus T_2 by disguising a set of polynomials over $\mathbb{F}_q$. Galbraith showed this to be weak; we present his argument in Section 9.4.4.

In §9.4.5 we suggest a novel way of disguising the same torus using the rational parameterisations of Rubin & Silverberg from §8.2. Finally we discuss why this does not securely disguise T_2, and make comments about higher–degree tori and their suitability for BBG cryptography.

9.4.1 Previous Research: Galbraith's Disguise of T_2

Galbraith in [28] described a method to disguise the simplest torus, $T_2(\mathbb{F}_{q^m})$. We present this now: Assume that $q = 2^s$ and that the extension degree m and s are both odd. Let some simple extension field be defined by $\mathbb{F}_{q^m} := \mathbb{F}_q(\alpha)$ where $\alpha \in \mathbb{F}_{q^m}$ is an algebraic element of order m. Let $\overline{\alpha}$ denote the Galois conjugate of α and let σ be the Frobenius automorphism of $\mathbb{F}_{q^m}$ that fixes $\mathbb{F}_q$. Then as usual one has $\alpha^2 + \alpha + 1 = 0$ and $\overline{\alpha} = \alpha + 1$.

For generic elements $a + b\alpha, c + d\alpha \in \mathbb{F}_{q^{2m}}^*$ one can define the group law for a projectively represented torus as:

Definition 9.4.1. The torus

$$T_2(\mathbb{F}_{q^m}) := \left\{ \frac{a + b\alpha}{a + b\overline{\alpha}} : a, b \in \mathbb{F}_{q^m}, (a, b) \neq (0, 0) \right\} \subset \mathbb{F}_{q^{2m}}^*$$

forms a group under the operation

$$(a + b\alpha)(c + d\alpha) = ac + bd + (ad + bc + bd)\alpha. \tag{9.4.1}$$

Here only the numerator is defined as it is understood that the denominator is simply its Galois conjugate.

We now use this projective representation of T_2 to construct a BBG.

9.4.2 Construction

Using equation (9.4.1) one can describe the T_2–group law in terms of m–tuples. One way is to construct a set of $2m$ quadratic homogenous polynomials in $4m$ variables

$$M_i(\underline{a}, \underline{b}, \underline{c}, \underline{d}) \qquad \text{for} \qquad i = 0, 1, \ldots, 2m-1$$

over $\mathbb{F}_q$. If $a \sim \underline{a} = (a_0, a_1, \ldots, a_{m-1})^{\mathrm{T}}$ and similarly for $b, c, d \in \mathbb{F}_q$ then the group law $e + f\alpha = ac + bd + (ad + bc + bd)\alpha$ is given by;

$$\begin{cases} e_i & = M_i(\underline{a}, \underline{b}, \underline{c}, \underline{d}) \quad \text{for} \quad 0 \leq i < m, \\ f_{i-m} & = M_i(\underline{a}, \underline{b}, \underline{c}, \underline{d}) \quad \text{for} \quad m \leq i < 2m. \end{cases} \tag{9.4.2}$$

Hence one can describe the T_2–group law on $\mathbb{F}_q^{2m}$ by a vector M of polynomials

$$\big(\underline{e}, \underline{f}\big) = \mathrm{M}\big(\underline{a}, \underline{b}, \underline{c}, \underline{d}\big) = \big(M_0(\underline{a}, \underline{b}, \underline{c}, \underline{d}), \ldots, M_{2m-1}(\underline{a}, \underline{b}, \underline{c}, \underline{d})\big) \tag{9.4.3}$$

whose elements lie over $\mathbb{F}_q$. This is similar to the treatment given in §9.2 whereby we now wish to disguise the natural representation $X = (\mathbb{F}_q^{2m}, \mathrm{M})$.

Let the disguising function $\psi : X \to Y$ be defined by generating a invertible linear transformation $\mathrm{U} \in \mathrm{GL}_{2m}(\mathbb{F}_q)$ in $2m$–dimensional space such that $\psi(\underline{a}, \underline{b}) = \mathrm{U}(a_0, \ldots, a_{m-1}, b_0, \ldots, b_{m-1})^{\mathrm{T}}$ $(= (\hat{\underline{a}}, \hat{\underline{b}}))$. Under this one has

$$\mathrm{U} : X = \big(\mathbb{F}_q^{2m}, \mathrm{M}\big) \to \big(\mathbb{A}^{2m}(\mathbb{F}_q), \mathrm{N}\big) = Y$$

where Y as usual denotes the disguised representation. The T_2–group law is then a system of $2m$–polynomials N which are computed from the set of equations (9.4.3) by

$$\mathrm{N}\big(\hat{\underline{a}}, \hat{\underline{b}}, \hat{\underline{c}}, \hat{\underline{d}}\big) = \mathrm{UM}\big(\mathrm{U}^{-1}(\hat{\underline{a}}, \hat{\underline{b}}), \mathrm{U}^{-1}(\hat{\underline{c}}, \hat{\underline{d}})\big) \tag{9.4.4}$$

where juxtaposition denotes normal matrix multiplication and $\mathrm{N}\big(\hat{\underline{a}}, \hat{\underline{b}}, \hat{\underline{c}}, \hat{\underline{d}}\big) = \big(\hat{\underline{e}}, \hat{\underline{f}}\big)$.

To compute with $Y = (\mathbb{A}^{2m}(\mathbb{F}_q), \mathrm{N})$, a BBG/public–user would simply generate four random m–tuples over $\mathbb{F}_q$ and use them to evaluate N. Thus the user is able to compute the T_2–group law of X (assuming the randomly chosen elements represent valid disguised elements) without directly using the natural representation of its elements; specifically the private basis $\{\mathfrak{B}, \alpha\}$ of $\mathbb{F}_q^{2m} \in X$.

***Remark* 9.4.2.** Clearly both q and m must be public, and so the disguised group order is known to be $q^m + 1$.

Corollary 9.4.3. *By construction, the algebraic structure $Y = (\mathbb{A}^{2m}(\mathbb{F}_q), \mathrm{N})$ which describes the T_2–group law on generic elements is a BBG under Definition 9.1.9.*

In fact, solely using Y it is not yet known whether one could compute inverses or test the equality of projective representations. What is now of principal interest however, is whether one can construct a method to un–disguise this representation of Y, computing a natural representation equivalent to X.

9.4.3 DHKA using Galbraith's Disguised T_2

Here we present how one could implement the Diffie–Hellman key agreement using Galbraith's T_2 BBG from Corollary 9.4.3 in §9.4.2:

***Remark* 9.4.4.** Let $(\underline{\hat{x}}, \underline{\hat{y}}) = \mathrm{U}(\underline{x}, \underline{y})$ and $\underline{P_1} = (\underline{\hat{x}}, \underline{\hat{y}})$. To compute the disguised representation of

$$[A](\underline{x}, \underline{y}) = \mathrm{U}^{-1}\underline{P_A}$$

one constructs the matrix $\mathrm{N}_d(\underline{P_r}) := \mathrm{N}(\underline{P_r}, \underline{P_r})$ which represents squaring. When $r \neq s$ one has $\underline{P_{r+s}} := \mathrm{N}(\underline{P_r}, \underline{P_s})$. Now one uses double–&–add algorithms with N, N_d and the binary representation of $A \in \mathbb{N}$ to evaluate P_A.

Initialisation: We canonically represent torus elements here only by their numerator. Fix a random torus element $x + y\alpha \in T_2(\mathbb{F}_{q^m})$ which has large prime order ℓ. Evaluate the $2m$–tuple $(\underline{x}, \underline{y}) \sim (x, y)$ and use the secret linear transformation U to compute the disguised $2m$–tuple $(\underline{\hat{x}}, \underline{\hat{y}}) = \mathrm{U}(\underline{x}, \underline{y})$. Now publish as the public parameters

$$\mathcal{V} := \{(\underline{\hat{x}}, \underline{\hat{y}}), \ell, Y\}$$

where $Y = \{q, m, \mathrm{N}\}$ is the disguised system.

Key Exchange:

1. Alice and Bob acquire $\mathcal{V}$ and then respectively choose random integers $A, B \in [1, \ell - 1] \subseteq \mathbb{N}$. These form their private–keys.
2. Using $\{\mathrm{N}, \mathrm{N}_d\}$ Alice computes her public–key $\underline{P_A}$ by the double–&–add method given in Remark 9.4.4.
3. Bob analogously computes his public–key $\underline{P_B}$.
4. As usual, they then publish/exchange their $2m$–tuple public keys, and Alice computes $\underline{P_{A+B}}$ from Bob's public key $\underline{P_B}$ and A.
5. Bob similarly evaluates a $\underline{P_{B+A}}$ from $\underline{P_A}$ and B.

Key Agreement: They now share a common secret $2m$–tuple $\underline{k} = \underline{P_{A+B}} = \underline{P_{B+A}}$. One then applies a key derivation function to $\underline{k}$ to obtain a useful cryptographic key.

9.4.4 Cryptanalysis of Galbraith's Construction

The construction of a BBG from T_2 in §9.4.2 allows us to describe the T_2–group law as a variety of $2m$ polynomials in $4m$ variables. This presentation however utilises no additional structure of the torus, and can be simply viewed as disguising a set of polynomials over a finite field.

This was shown to be weak by Galbraith in [28] whereby using the description $\mathrm{N} \in Y$ of the disguised group law, one is able to recover a natural representation of the underlying field $\mathbb{F}_{q^m}$. With this one can then recover a natural representation for all $2m$–tuples which represent elements of T_2 in the form $a + b\alpha \in T_2(\mathbb{F}_{q^m})$. Hence, an isomorphic representation and the structure of the original 'disguised' torus is recovered. We present this attack now:

Algorithm 9.4.1 (Attack of Galbraith's Disguised T_2)**.**

1. Generate a random $2m$–tuple $(\underline{a}, \underline{b})$. Due to Remark 9.4.2, the order q^m+1 of T_2 is known. Thus, one can compute a

$$\underline{\hat{w}} = (\underline{\hat{x}}, \underline{\hat{y}}) = (\underline{a}, \underline{b})^{q^m+1}$$

   by using the group operation N and any square–and–multiply algorithm.

2. Now $\underline{\hat{w}}$ is one of the (possibly many) disguised representations for the identity element of T_2; that is an un–disguised element of the form

$$\mathrm{U}^{-1}\underline{\hat{w}} = (\underline{x}, \underline{0}) = \underline{w}.$$

3. Hence using $\mathrm{N} \in Y$ one can now compute

$$\mathrm{N}(\underline{\hat{w}}, \underline{\hat{w}}) = \mathrm{N}(\underline{\hat{x}}, \underline{\hat{y}}, \underline{\hat{x}}, \underline{\hat{y}}) = \mathrm{U}\big((\underline{x^2} + \underline{0} \cdot \underline{0}), (\underline{x} \cdot \underline{0} + \underline{0} \cdot \underline{x} + \underline{0} \cdot \underline{0})\big) = \underline{\hat{w^2}}.$$

   Continuing iteratively one can construct the set of $m+1$ elements

$$\Delta = \{\mathrm{U}\underline{w}, \mathrm{U}\underline{w^2}, \mathrm{U}\underline{w^3}, \ldots, \mathrm{U}\underline{w^{m+1}}\}.$$

4. One now continues to find a defining polynomial for $\mathbb{F}_{q^m}$ as we did in §9.2.2. With a high probability we can find a polynomial–basis for $\mathbb{F}_{q^m}$. When this is unsuccessful, we return to `STEP 1`.

5. One now applies a linear transformation U_1 on the polynomials in N which diagonalises the first m variables. This ensures that the first m components correspond to $\mathbb{F}_{q^m}$ with the polynomial–basis $\{1, w, \ldots, w^{m-1}\}$.

6. One now recovers the torus structure: Generate a random m–tuple $\underline{y}$ and use this to construct the $2m$–tuple

$$\underline{\hat{u}} = (\underline{0}, \underline{y}).$$

   We do not have an especially useful form as we had in `STEP 3`, and so this just corresponds to an arbitrary un–disguised element $u = c + d\alpha$ for $c, d \in \mathbb{F}_{q^m}$.

7. Hence one computes the set

$$\Lambda = \{\mathrm{N}(\underline{\hat{w}^i}, \underline{\hat{u}}) \mid i = 1, \ldots, m\} = \{\underline{\hat{w}} \cdot \underline{\hat{u}}, \ldots, \underline{\hat{w^m}} \cdot \underline{\hat{u}}\}$$

   where $\underline{\hat{w^i}} \cdot \underline{\hat{u}}$ correspond to some element $c_i' + d_i'\alpha$. If all of the elements of Λ are of the form $(\underline{0}, \underline{y})$ we are done. Otherwise we compute a linear transformation U_2 such that all elements of Λ are in this form and then we set $\underline{\hat{u}} = \mathrm{U}_2\underline{\hat{u}}$.

8. Now $\underline{\hat{u}}$ is a 'purely quadratic' element and so we set $\alpha \sim \underline{\hat{u}}$ and by using the basis of $\mathbb{F}_{q^m}$ we computed in `STEP 5` we compute this α. We now have a complete basis for $\mathbb{F}_{q^{2m}}$ and so a natural representation X' of the torus T_2 is recovered.

9. One now uses §9.1.4 to compute a map between $Y \to X'$ so that particular cryptographic problem may be attacked using this method.

***Remark* 9.4.5.** This attack works as in `STEP 3` we are able to construct a simple field multiplication algorithm from the T_2–group description N, by essentially computing different disguised representations of the trivial operation

$$1 \cdot 1 = 1.$$

Since our $\underline{\hat{w}}$ was an identity element of the torus but not the field, this allowed us to create a polynomial–basis for $\mathbb{F}_{q^m}$ as we did in §9.2.2.

Thus our BBG operation became a black–box for field multiplication, and this is why it failed in an analogous way to simply disguised finite fields. If one wanted, one could now use a 'nicer' basis by computing a suitable isomorphism with Lenstra's method [47].

9.4.5 Disguising T_2 — A New Approach

Galbraith's disguise of T_2 failed as one was able to create an algorithm for field multiplication from the disguised T_2 group law $\mathrm{N} \in Y$.

We now describe an original and novel way of presenting a disguised group law of T_2 using Rubin & Silverberg's parameterisation from §8.2. This is achieved by publishing a set of matrices, whereby one computes the disguised T_2–group law by solving a system of linear equations rather than by direct substitution. Following this parameterisation we describe our new method in the sequel.

9.4.6 Construction

We will attempt to indirectly construct a T_2 BBG by disguising the partial group law defined on $\mathbb{A}^1(\mathbb{F}_{q^m})$ given in §8.2. Here we detail this for a non–binary field since the treatment for the binary case is completely analogous:

We need to disguise the partial T_2 group law on $\mathbb{A}^1$ given by equation (8.2.6)

$$a \star b = \frac{ab + d}{a + b}.$$

The division on the right hand side is a problem for our methodology, since it cannot be computed by evaluating a multivariate polynomial over $\mathbb{F}_q$ of low degree. Thus, we set $c = a \star b$ and reformulate this equation to give

$$(a + b)c = ab + d. \tag{9.4.5}$$

We now can use this to describe the partial group law. One way is to construct two matrices; an $m \times m$ matrix M whose entries are linear polynomials and an

$m \times 1$ vector N whose entries are quadratic polynomials, both of which are in $2m$ variables containing elements of the form

$$\left.\begin{matrix} M_{ij}(\underline{a},\underline{b}) \\ N_i(\underline{a},\underline{b}) \end{matrix}\right\} \qquad \text{for} \qquad i,j = 0,1,\ldots,m-1$$

over $\mathbb{F}_q$. Under this the partial group law is given by

$$\overbrace{\mathrm{M}(\underline{a},\underline{b})\underline{c}}^{\sim c(a+b)} = \underbrace{\mathrm{N}(\underline{a},\underline{b})}_{\sim ab+d} \tag{9.4.6}$$

where juxtaposition denotes normal matrix multiplication and

$$\mathrm{M} := \big(M_{i,j}(\underline{a},\underline{b})\big)_{0\le i,j\le m-1}, \qquad \mathrm{N} := \big(N_i(\underline{a},\underline{b})\big)_{0\le i\le m-1}$$

***Remark* 9.4.6.** In order to compute the group law (and hence c), one generates two m–tuples $\underline{a},\underline{b}$ and uses these to evaluate the matrices M and N. One now solves the linear algebra problem

$$\begin{pmatrix} m_{0,0} & \cdots & m_{0,m-1} \\ \vdots & \ddots & \vdots \\ m_{m-1,0} & \cdots & m_{m-1,m-1} \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} n_0 \\ \vdots \\ n_{m-1} \end{pmatrix}$$

where only the $c_i \in \mathbb{F}_q$ are unknown with $m_{i,j} = M_{i,j}(\underline{a},\underline{b})$ and $n_i = N_i(\underline{a},\underline{b})$.

Upon computing the solution to this system of linear equations, $\underline{c}$, one has computed the group operation on m–tuples as required.

We wish to disguise this natural representation $X = \big(\mathbb{F}_q^m, \{\mathrm{M},\mathrm{N}\}\big)$: Let the disguising function $\psi : X \to Y$ be defined by generating a random invertible linear transformation $\mathrm{U} \in \mathrm{GL}_m(\mathbb{F}_q)$ in m–dimensional space and by defining $\psi(\underline{a}) = \mathrm{U}(a_0,\ldots,a_{m-1})^{\mathrm{T}} = \hat{\underline{a}}$. Under this one has

$$\mathrm{U} : X = \big(\mathbb{F}_q^m, \{\mathrm{M},\mathrm{N}\}\big) \to \big(\mathbb{A}^m(\mathbb{F}_q), \{\mathrm{M}',\mathrm{N}'\}\big) = Y$$

where Y as usual denotes the disguised representation. The disguised partial group law is then constructed from the matrices given in equation (9.4.6) as follows:

$$\begin{aligned} \mathrm{UM}(\mathrm{U}^{-1}\hat{\underline{a}},\mathrm{U}^{-1}\hat{\underline{b}})\mathrm{U}^{-1}\hat{\underline{c}} = \mathrm{UN}(\mathrm{U}^{-1}\hat{\underline{a}},\mathrm{U}^{-1}\hat{\underline{b}}) &\iff \\ \mathrm{M}'(\hat{\underline{a}},\hat{\underline{b}})\hat{\underline{c}} = \mathrm{N}'(\hat{\underline{a}},\hat{\underline{b}}) & \end{aligned} \tag{9.4.7}$$

with the two relations

$$\begin{aligned} \mathrm{M}'(\hat{\underline{a}},\hat{\underline{b}}) &= \mathrm{UM}(\mathrm{U}^{-1}\hat{\underline{a}},\mathrm{U}^{-1}\hat{\underline{b}})\mathrm{U}^{-1}, \\ \mathrm{N}'(\hat{\underline{a}},\hat{\underline{b}}) &= \mathrm{UN}(\mathrm{U}^{-1}\hat{\underline{a}},\mathrm{U}^{-1}\hat{\underline{b}}). \end{aligned}$$

These matrices which describe the disguised group law are then published. Furthermore, it is immediate that using two disguised m–tuples $\hat{\underline{x}},\hat{\underline{y}}$, a public user may compute the disguised partial group law using an analogous method of that described in Remark 9.4.6. Thus we have now constructed a BBG with which to compute with representatives of elements of $T_2(\mathbb{F}_{q^m})$.

Corollary 9.4.7. *Under Definition 9.1.9 one now has a candidate BBG in the algebraic structure* $Y = (\mathbb{A}^m(\mathbb{F}_q), \{\mathrm{M}',\mathrm{N}'\})$ *which describes the partial group law on* $\mathbb{A}^1(\mathbb{F}_{q^m})$ *representing the group law of* $T_2(\mathbb{F}_{q^m})$.

9.4.7 Examples

For ease of exposition, we represent the disguised m–tuples via

$$\begin{aligned}\hat{\underline{a}} &= \mathrm{U}\underline{a} = \mathrm{U}(a_0, a_1, \ldots, a_{m-1}) = (x_0, x_1, \ldots, x_{m-1}),\\ \hat{\underline{b}} &= \mathrm{U}\underline{b} = \mathrm{U}(b_0, b_1, \ldots, b_{m-1}) = (y_0, y_1, \ldots, y_{m-1}),\\ \hat{\underline{c}} &= \mathrm{U}\underline{c} = \mathrm{U}(c_0, c_1, \ldots, c_{m-1}) = (z_0, z_1, \ldots, z_{m-1}).\end{aligned}$$

Using this we now present an example of the creation of a disguised system and an example of computing with one:

***Example* 9.4.8** (Creating a Disguised System)**.** Let $\mathbb{F}_q = \mathbb{F}_{17}$, $m = 3$ and w be a root of the irreducible degree 3 polynomial $x^3 + x + 14$ over $\mathbb{F}_q$ such that $\mathbb{F}_{q^m} := \mathbb{F}_q(w)$.

Here $d = 6w^2 + 10w + 3$ is a random non–square element of $\mathbb{F}_{q^m}^*$, hence set $\mathbb{F}_{q^{2m}} := \mathbb{F}_{q^m}(\sqrt{d}) = \mathbb{F}_q(\sqrt{d})(w)$. The maps σ, ψ and ρ are all defined as in §8.2.

The private user now computes the two undisguised matrices M and N using equation (9.4.6):

$$\mathrm{M} = \begin{pmatrix} a_0 + b_0 & a_1 + b_1 & a_2 + b_2 \\ 3a_2 + 3b_2 & a_0 + 16a_2 + b_0 + 16b_2 & a_1 + b_1 \\ 3a_1 + 3b_1 & 16a_1 + 3a_2 + 16b_1 + 3b_2 & a_0 + 16a_2 + b_0 + 16b_2 \end{pmatrix},$$

$$\mathrm{N} = \begin{pmatrix} a_0b_0 + 3a_1b_2 + 3a_2b_1 + d_0 \\ a_0b_1 + a_1b_0 + 16a_1b_2 + 16a_2b_1 + 3a_2b_2 + d_1 \\ a_0b_2 + a_1b_1 + a_2b_0 + 16a_2b_2 + d_2 \end{pmatrix}$$

and generates a random secret invertible linear transformation (the trapdoor)

$$\mathrm{U} = \begin{pmatrix} 12 & 1 & 15 \\ 8 & 12 & 1 \\ 10 & 16 & 5 \end{pmatrix} \in GL_3(\mathbb{F}_q).$$

Now one computes the two disguised matrices $\mathrm{M}'(\hat{\underline{a}}, \hat{\underline{b}}) = \{M_0, M_1, M_2\}$ where

$$\begin{aligned}{M_0}^{\mathrm{T}} &:= \begin{pmatrix} 12x_0 + x_1 + 15x_2 + 12y_0 + y_1 + 15y_2 \\ 5x_0y_0 + 2x_0y_1 + 10x_0y_2 + 5x_0 + 2x_1 + 10x_2 + 5y_0 + 2y_1 + 10y_2 \\ 10x_0 + 4x_1 + 16x_2 + 10y_0 + 4y_1 + 16y_2 \end{pmatrix}\\ {M_1}^{\mathrm{T}} &:= \begin{pmatrix} x_0 + 3x_1 + 10x_2 + y_0 + 3y_1 + 10y_2 \\ 2x_0 + 13x_1 + 9x_2 + 2y_0 + 13y_1 + 9y_2 \\ 4x_0 + 10x_1 + 3x_2 + 4y_0 + 10y_1 + 3y_2 \end{pmatrix}\\ {M_2}^{\mathrm{T}} &:= \begin{pmatrix} 15x_0 + 10x_1 + 15x_2 + 15y_0 + 10y_1 + 15y_2 \\ 10x_0 + 9x_1 + 6x_2 + 10y_0 + 9y_1 + 6y_2 \\ 16x_0 + 3x_1 + 4x_2 + 16y_0 + 3y_1 + 4y_2 \end{pmatrix}\end{aligned}$$

and an $\mathrm{N}'(\hat{\underline{a}}, \hat{\underline{b}})$ given by

$$\begin{pmatrix} 12x_0y_0 + x_0y_1 + 15x_0y_2 + x_1y_0 + 3x_1y_1 + 10x_1y_2 + 15x_2y_0 + 10x_2y_1 + 15x_2y_2 \\ 5x_0y_0 + 2x_0y_1 + 10x_0y_2 + 2x_1y_0 + 13x_1y_1 + 9x_1y_2 + 10x_2y_0 + 9x_2y_1 + 6x_2y_2 + 1 \\ 10x_0y_0 + 4x_0y_1 + 16x_0y_2 + 4x_1y_0 + 10x_1y_1 + 3x_1y_2 + 16x_2y_0 + 3x_2y_1 + 4x_2y_2 \end{pmatrix}.$$

Then $Y = \{q, m, \mathrm{M}', \mathrm{N}'\}$ is published and describes the BBG.

Using Example 9.4.8 we now demonstrate how one computes with disguised representations.

***Example* 9.4.9** (Computing With a Disguised System)**.** The public user acquires Y and using two disguised input values (here chosen at random) $\hat{\underline{a}} = (3,1,16)^{\mathrm{T}}$ and $\hat{\underline{b}} = (10,14,5)^{\mathrm{T}}$ evaluates M', N' to get

$$\mathrm{M}'(\hat{\underline{a}}, \hat{\underline{b}}) = \begin{pmatrix} 10 & 16 & 16 \\ 13 & 2 & 10 \\ 14 & 0 & 14 \end{pmatrix} \qquad \text{and} \qquad \mathrm{N}'(\hat{\underline{a}}, \hat{\underline{b}}) = \begin{pmatrix} 7 \\ 2 \\ 1 \end{pmatrix}.$$

Solving the linear equation

$$\mathrm{M}'(\hat{\underline{a}}, \hat{\underline{b}})\hat{\underline{c}} = \mathrm{N}'(\hat{\underline{a}}, \hat{\underline{b}})$$

over $\mathbb{F}_q$ gives the unique solution $\hat{\underline{c}} = (5,12,4)^{\mathrm{T}}$. Thus the public user has computed the group law using only disguised elements.

This corresponds to the private user having elements

$$\begin{aligned} \underline{a} &= \mathrm{U}^{-1}\hat{\underline{a}} = (10,11,5)^{\mathrm{T}} \sim 5w^2 + 11w + 10, \\ \underline{b} &= \mathrm{U}^{-1}\hat{\underline{b}} \sim 16w^2 + 13w + 10 \end{aligned}$$

and evaluating in $\mathbb{F}_{q^m} = \mathbb{F}(w)(\sqrt{d})$

$$c = (ab + d)/(a + b) = 16w^2 + 8w + 15 \sim \mathrm{U}^{-1}\hat{\underline{c}}$$

and so it is well–defined.

9.4.8 Cryptanalysis of Our Disguised T_2

Unfortunately, the BBG presented in §9.4.6 is easily un–disguised. The attack utilises the fact that a cryptanalyst may use the published matrices to compute other things than that they were intended for. We present this attack now:

Algorithm 9.4.2 (Attack of Our Disguised T_2)**.**

1. Eve acquires the public $\mathcal{V}$ and picks a random m–tuple $\hat{\underline{w}}$ which with a high degree of probability corresponds to some un–disguised (and unknown to the attacker) $\underline{w}$

2. As $\mathrm{M}'(\hat{\underline{a}}, \hat{\underline{b}})$ and $\mathrm{N}'(\hat{\underline{a}}, \hat{\underline{b}})$ are just matrices, the attacker sets $\hat{\underline{a}} = \hat{\underline{w}}$, $\hat{\underline{b}} = \underline{0}$ and $\hat{\underline{c}} = \hat{\underline{w}}$ and then computes (whilst ignoring equation (9.4.7))

   $$\mathrm{M}'(\hat{\underline{a}}, \hat{\underline{b}})\hat{\underline{c}} = \mathrm{M}'(\hat{\underline{w}}, \underline{0})\hat{\underline{w}} = \underline{(\hat{w^2})} = \mathrm{U}\underline{(w^2)}$$

   which holds under the construction of equation (9.4.7).

3. Setting $\hat{\underline{c}} = \underline{(\hat{w^2})}$ and continuing iteratively allows the attacker to compute the $m+1$ blinded representations

   $$\Delta = \{\mathrm{U}\underline{w}, \mathrm{U}\underline{w^2}, \mathrm{U}\underline{w^3}, \ldots, \mathrm{U}\underline{w^{m+1}}\}.$$

   We can now continue to find a defining polynomial for $\mathbb{F}_{q^m}$ as we did in §9.2.2. With a high probability we can now find a polynomial basis for $\mathbb{F}_{q^m}$.

4. To recover a non–square element, the attacker sets $\hat{\underline{a}} = \underline{0}$ and then trivially has
$$\mathrm{N}'(\underline{0}, \hat{\underline{b}}) = \hat{\underline{d}}.$$

5. Using §9.1.4 one constructs an isomorphism to decompose the disguised elements in terms of our new basis and that of $\hat{\underline{d}}$. Using this one can compute with the original partial group law in equation (8.2.6) and we are done.

9.4.9 Disguising Higher–Degree Tori

It is natural to consider higher dimensional tori for disguising such as T_6 and T_{30} and particularly those with *rational parameterisations*: birational maps $\rho : T_n \to \mathbb{A}^{\phi(n)}$ defined over $\mathbb{F}_q$. Let us consider the torus T_6:

$$T_6(\mathbb{F}_{q^m}) = \{\alpha \in \mathbb{F}_{q^{6m}} \mid \mathrm{N}_{\mathbb{F}_{q^{6m}}/\mathbb{F}_{q^{3m}}}(\alpha) = 1, \mathrm{N}_{\mathbb{F}_{q^{6m}}/\mathbb{F}_{q^{2m}}}(\alpha) = 1\},$$

where T_6 is of dimension 2 and $\#T_6(\mathbb{F}_{q^m}) = \Phi_6(q^m) = q^{2m} - q^m + 1$ for the cyclotomic polynomial $\Phi_d(x)$.

Rubin & Silverberg in [67] presented a cryptosystem based on the rational parameterisation of the torus T_6, called CEILIDH. This works similarly to XTR and LUC, however what was important is they showed one could define a rational parameterisation (a birational map)

$$\psi : \mathbb{A}^2(\mathbb{F}_{q^m}) \dashrightarrow T_6(\mathbb{F}_{q^m}).$$

We use $\dashrightarrow$ since not all points can be mapped, for details see ([67], p8).

As an example of this; setting $q^m \equiv 2, 5 \pmod 9$, $x = \zeta_3$ and $y = \zeta_9 + \zeta_9^{-1}$ they showed that when one defines $\mathbb{F}_{q^{3m}} = \mathbb{F}_{q^m}[y]$ and $\mathbb{F}_{q^{6m}} = \mathbb{F}_{q^{3m}}[x]$, one has

$$\psi(\alpha_1, \alpha_2) = \frac{1 + \alpha_1 y + \alpha_2(y^2 - 2) + (1 - \alpha_1^2 - \alpha_2^2 + \alpha_1\alpha_2)x}{1 + \alpha_1 y + \alpha_2(y^2 - 2) + (1 - \alpha_1^2 - \alpha_2^2 + \alpha_1\alpha_2)x^2}. \tag{9.4.8}$$

The details of this parameterisation is beyond the scope of this thesis, however Rubin & Silverberg introduced these maps to enable element compression. Granger et al. in ([34], p10) considered efficient arithmetic on tori. They argued that the best performance one could have using T_6 was via mapping it 'up' to T_2, computing the group law there and then mapping the result back 'down' to T_6. This is made possible because $|T_6(\mathbb{F}_q)| = (q^2 - q + 1)|(q^3 + 1) = |T_2(\mathbb{F}_{q^3})|$. Their results show that disguising T_6 is uninteresting from a practical point–of–view since one has already discounted using T_2. Similar arguments can be applied to higher dimensional tori, and we do not consider them in our work here.

9.5 Disguising Trace–Based Methods: LUC

Our cryptanalysis has already shown that it is not secure or practical to disguise finite fields and algebraic tori. We now turn our attention to whether the trace–based methods of LUC and later in §8.4 XTR, make suitable BBG candidates.

Motivation 9.5.1. We consider LUC as a possible object for disguising DL–cryptography since it has less arithmetical structure than did our undisguised torus in §9.4.5: Although exponentiation is well defined since $T(g^a) = T(g^{-a})$ and from Corollary 8.3.3, easy to compute, multiplication is not since generally $T(gh) \neq T(gh^{-1})$ as was shown in Remark 8.3.4.

As §8.3 details, when we use LUC we actually work with traces over the ground field which represent elements of the group $G_{2,q}$, rather than the torus elements themselves. Hence it is worth noting that this indirect relationship does not natively form a BBG:

Corollary 9.5.2. *Standard LUC given in §8.3 does not form a BBG.*

Proof. We show that nothing is 'disguised' here: One can find a non–square $a \in \mathbb{F}_{q^m}$ and use this to define $\mathbb{F}_{q^{2m}} = \mathbb{F}_{q^m}(\sqrt{a})$. Given t_1 and t_n one wants to compute $g, h \in \mathbb{F}_{q^{2m}}$ such that $\mathrm{Tr}(g) = t_1$ and $\mathrm{Tr}(h) = t_n$. Noting that $(X - g)(X - \sigma(g)) = X^2 - T(g) + 1$, one can now solve and recover a g up to conjugate. One can analogously compute h up to conjugate. Hence we trivially have the non–black–box structure of the group. □

9.5.1 Construction

We need to disguise the trace representation of LUC. The sequence $\{t_n\}$ defined in Corollary 8.3.3 works solely in $\mathbb{F}_{q^m}$. Hence using equation (8.3.5) one can describe the LUC group law in terms of m–tuples. One way is to construct a set of m polynomials in $3m$ variables which describe the Lucas recursion formula given in equation (8.3.5) as follows:

$$\underline{t_n} = \overbrace{\mathrm{A}(\underline{t_1}, \underline{t_{n-1}})}^{\sim t_1 \cdot t_{n-1}} + \underline{t_{n-2}} = \mathrm{M}\big(\underline{t_1}, \underline{t_{n-1}}, \underline{t_{n-2}}\big) \tag{9.5.1}$$

where A and M are vectors of dimension m with the polynomial elements

$$\underline{(t_n)_i} := M_i\big(\underline{t_1}, \underline{t_{n-1}}, \underline{t_{n-2}}\big)$$

defined over $\mathbb{F}_q$. Here one has the corresponding initial conditions; $\underline{t_0} \sim 2$ and $\underline{t_1} \sim T(g^a)$. It is this natural representation $X = \big(\mathbb{F}_q^m, \{\mathrm{M}, \mathrm{I}_0\}\big)$ which we wish to disguise where I_0 denotes the initial conditions $\{\underline{t_0}, \underline{t_1}\}$.

Let the disguising function $\psi : X \to Y$ be defined by generating a random invertible $\mathrm{U} \in \mathrm{GL}_m(\mathbb{F}_q)$ such that $\psi(\underline{a}) = \mathrm{U}\underline{a} = \underline{\hat{a}}$. Under this one has

$$\mathrm{U} : X = \big(\mathbb{F}_q^m, \{\mathrm{M}, \mathrm{I}_0\}\big) \to \big(\mathbb{A}^m(\mathbb{F}_q), \{\mathrm{N}, \mathrm{I}\}\big) = Y$$

where Y as usual denotes the disguised representation. Disguising in this way is well–defined since the trace is trivially $\mathbb{F}_{q^m}$–linear since $\mathrm{Tr}(g+h) = \mathrm{Tr}(g)+\mathrm{Tr}(h)$ and $\mathrm{Tr}(c \cdot g) = c \cdot \mathrm{Tr}(g)$ for all $c \in \mathbb{F}_{q^m}$.

The disguised group law $\mathrm{N} = \mathrm{UM}$ is then a system of m polynomials which are computed from the set of equations in (9.5.1) by:

$$\begin{aligned} \mathrm{U}^{-1}\underline{\hat{t_n}} = \mathrm{UM}\big(\mathrm{U}^{-1}\underline{\hat{t_1}}, \mathrm{U}^{-1}\underline{\hat{t_{n-1}}}, \mathrm{U}^{-1}\underline{\hat{t_{n-2}}}\big) \Leftrightarrow \\ \underline{\hat{t_n}} = \mathrm{N}\big(\underline{\hat{t_1}}, \underline{\hat{t_{n-1}}}, \underline{\hat{t_{n-2}}}\big) \end{aligned} \tag{9.5.2}$$

for a public vector N of dimension m with polynomial elements over $\mathbb{F}_q$. Similarly now one computes the disguised initial conditions: $\mathrm{I} = \{\mathrm{U}\underline{t_0}, \mathrm{U}\underline{t_1}\}$.

Corollary 9.5.3. *Under Definition 9.1.9 one now has a candidate BBG in the algebraic structure* $Y = (\mathbb{A}^m(\mathbb{F}_q), \{\mathrm{N}, \mathrm{I}\})$ *which describes the LUC group operation on disguised trace elements.*

To compute with $Y = \big(\mathbb{A}^m(\mathbb{F}_q), \mathrm{N}\big)$, a public/BBG–user acquires Y and using an integer a and an m–tuple $\underline{\hat{x}}$ over $\mathbb{F}_q$ iteratively evaluates

$$\begin{aligned} \underline{\hat{t_2}} &= \mathrm{N}\big(\underline{\hat{x}}, \underline{\hat{t_1}}, \underline{\hat{t_0}}\big) \\ \vdots \qquad & \qquad \vdots \\ \underline{\hat{t_a}} &= \mathrm{N}\big(\underline{\hat{x}}, \underline{\hat{t_{a-1}}}, \underline{\hat{t_{a-2}}}\big) \end{aligned}$$

Thus the user is able to compute the disguised group law of X without directly using its natural representation; specifically the basis $\mathcal{B}$ of $\mathbb{F}_q^m \in X$.

Since U is linear, such computations are efficient as the elements of N are quadratic polynomials over $\mathbb{F}_q$.

***Remark* 9.5.4.** Clearly if t_1 (and hence $\underline{\hat{t_1}}$) were fixed for all computations, then one could publish a simplified disguised group law description

$$\underline{\hat{t_n}} = \mathrm{N}\big(\underline{\hat{t_{n-1}}}, \underline{\hat{t_{n-2}}}\big).$$

9.5.2 Cryptanalysis of Disguised LUC

The BBG presented in §9.5.1 can be undisguised. The goal of the cryptanalyst here is analogous to before: Given the disguised representation of LUC, obtain a representation for the field $\mathbb{F}_{q^m}$. Then pull the disguised trace values $\underline{\hat{t_i}}$ back to elements over our newly computed natural representation for $\mathbb{F}_{q^m}$.

Eve similar to before does not use N to compute the trace at all, but manipulates it to give her a finite field multiplication algorithm. We first present a lemma which will aid in the attack:

Lemma 9.5.5. *One can create a disguised field multiplication algorithm* $\mathcal{A}$ *for* $\mathbb{F}_{q^m}$ *from the disguised group description given in* Y.

Proof. Set the multiplication algorithm $\mathcal{A}$ to be defined by

$$\mathcal{A}\big(\underline{\hat{x}}, \underline{\hat{y}}\big) := \mathrm{N}\big(\underline{\hat{x}}, \underline{\hat{y}}, \underline{0}\big)$$

for two arbitrary m–tuples $\underline{\hat{x}}$ and $\underline{\hat{y}}$. Then clearly $\mathcal{A}$ is a disguised multiplication algorithm for $\mathbb{F}_{q^m}$ since from equation (9.5.1),

$$\mathrm{U}^{-1}\mathcal{A}\big(\underline{\hat{x}}, \underline{\hat{y}}\big) = \mathrm{M}\big(\mathrm{U}^{-1}\underline{\hat{x}}, \mathrm{U}^{-1}\underline{\hat{y}}, \mathrm{U}^{-1}\underline{0}\big) \sim xy \in \mathbb{F}_{q^m}.$$

□

Algorithm 9.5.1 (Attack of Disguised LUC)**.**

1. Eve acquires Y and picks a random $2m$–tuple $\underline{\hat{w}}$ which with a high degree of probability corresponds to some undisguised (and unknown to the attacker) $\underline{w}$. Clearly, one could pick $\underline{\hat{w}} = \underline{\hat{t_1}}$ which is known to correspond to the undisguised t_1.

2. Using Lemma 9.5.5 Eve constructs a disguised field multiplication algorithm $\mathcal{A}$ and use this to recursively compute the set

$$\Delta = \{\underline{\hat{w_1}}, \underline{\hat{w_2}}, \ldots, \underline{\hat{w_{m+1}}}\} = \{\mathrm{U}\underline{w}, \mathrm{U}\underline{w^2}, \mathrm{U}\underline{w^3}, \ldots, \mathrm{U}\underline{w^{m+1}}\}$$

by iteratively evaluating $\underline{\hat{w_i}} = \mathcal{A}(\underline{\hat{w_{i-1}}}, \underline{\hat{w}})$.

3. One now has $m+1$ vectors in an m–dimensional vector space, so one can continue to find a defining polynomial for $\mathbb{F}_{q^m}$ as we did in §9.2.2. With a high probability one can now find a polynomial basis for $\mathbb{F}_{q^m}$.

4. Now one proceeds as before and computes a map between the disguised representation Y and our newly computed natural one X' using §9.1.4.

***Remark* 9.5.6.** As previously mentioned, using traces one cannot distinguish between an element and its conjugates. However, one could map this problem into a standard DLP in group $G_{2,q}$ by finding the correct root of

$$(x-g)(x-\sigma(g)) = x^2 - \mathrm{Tr}(g)x + 1 = 0.$$

This would then yield g and its conjugate from $\mathrm{Tr}(g)$. One would similarly compute a pair of values for $\mathrm{Tr}(g^n)$.

9.6 Disguising Trace–Based Methods: XTR

We wish to see if using XTR from §8.4 allows one to create a BBG under disguising. XTR itself is a natural extension of LUC, although it has a more complex presentation. Due to this similarity however, one expects it to fail in a similar manner as did LUC. We now present the details.

9.6.1 Construction

We consider XTR as a possible object for disguising since it has less arithmetical structure than did our undisguised torus given in §9.4.5. Explicitly; although exponentiation is well defined since $T(g^a) = T(g^{a(p-1)}) = T(g^{-ap})$, multiplication is not since generally by Remark 8.3.4,

$$T(gh) \neq T(g^{p-1}h) \neq T(g^{-p}h).$$

To describe XTR, one needs $\mathbb{F}_{p^2}$, the initial conditions $\{t_0, t_1\}$, a description of the third–order relation t_{u+v} and its four identities from Lemma 8.4.4 which we (respectively) name I, II, III & IV from which one can derive t_2.

Setting $\mathbb{F}_{q^m} = \mathbb{F}_p$, everything in §8.4 still holds, hence using Lemma 8.4.4 one can now describe the XTR group law in terms of $2m$–tuples where the $n < 0$ case is dealt with by noting $t_{-n} = t_n^{q^m}$:

Consider Identity II from Lemma 8.4.4: Clearly one can rewrite t_{n+2} using $2m$–tuples as

$$\begin{aligned} \underline{t_{n+2}} &= \overbrace{\mathrm{M}(\underline{t_1}, \underline{t_{n+1}})}^{\sim t_1 \cdot t_{n+1}} - \overbrace{\mathrm{N}(\underline{t_1}, \underline{t_n})}^{\sim t_n (t_1)^{q^m}} + \underline{t_{n-1}} \\ &= \mathrm{R_{II}}(\underline{t_1}, \underline{t_{n-1}}, \underline{t_n}, \underline{t_{n+1}}) \end{aligned} \tag{9.6.1}$$

where M, N and $\mathrm{R_{II}}$ are dimension $2m$ vectors with the polynomial elements

$$\underline{(t_{n+2})_i} = R_i\big(\underline{t_1}, \underline{t_{n-1}}, \underline{t_n}, \underline{t_{n+1}}\big)$$

defined over $\mathbb{F}_q$. Analogously one then computes the following matrices

$$\begin{aligned} \underline{t_{2n}} &= \mathrm{R_I}\ \ \big(\underline{t_n}\big), & \underline{t_{2n-1}} &= \mathrm{R_{III}}\big(\underline{t_1}, \underline{t_{n-1}}, \underline{t_n}, \underline{t_{n+1}}\big), \\ \underline{t_{2n+1}} &= \mathrm{R_{IV}}\big(\underline{t_1}, \underline{t_{n-1}}, \underline{t_n}, \underline{t_{n+1}}\big), & \underline{t_{u+v}} &= \mathrm{R}\quad \big(\underline{t_{u-2v}}, \underline{t_{u-v}}, \underline{t_u}, \underline{t_v}\big). \end{aligned}$$

It is this natural representation $X = (\mathbb{F}_q^{2m}, \{\mathcal{V}, \mathrm{I_0}\})$ which we wish to disguise where $\mathrm{I_0}$ denotes the initial conditions $\{\underline{t_0}, \underline{t_1}\}$ and $\mathcal{V}$ is the collection of rules $\{\mathrm{R}, \mathrm{R_I}, \mathrm{R_{II}}, \mathrm{R_{III}}, \mathrm{R_{IV}}\}$.

Let the disguising function $\psi : X \to Y$ be defined by generating a random invertible $\mathrm{U} \in \mathrm{GL}_{2m}(\mathbb{F}_q)$ such that $\psi(\underline{a}) = \mathrm{U}\underline{a} = \underline{\hat{a}}$. Under this one has

$$\mathrm{U} : X = \big(\mathbb{F}_q^{2m}, \{\mathcal{V}, \mathrm{I_0}\}\big) \to \big(\mathbb{A}^{2m}(\mathbb{F}_q), \{\mathcal{W}, \mathrm{I}\}\big) = Y$$

where Y as usual denotes the disguised representation. Disguising in this way is well defined since the trace is $\mathbb{F}_{q^{2m}}$–linear since: $T(g+h) = T(g) + T(h)$ and $T(\alpha \cdot h) = \alpha T(h)$ for all $\alpha \in \mathbb{F}_q$ and $g, h \in \mathbb{F}_{q^{6m}}^*$.

The disguised group law is then a system $\mathcal{W}$ of five sets of $2m$ polynomials which are computed from $\mathcal{V}$. We explicitly detail how this is done for $\mathrm{R_{II}}$ from equation (9.6.1). The disguised description of $\mathrm{R_{II}}$ is computed by:

$$\begin{aligned} \mathrm{U}^{-1}(\underline{\hat{t_{n+2}}}) = \mathrm{UR_{II}}\big(\mathrm{U}^{-1}\underline{\hat{t_1}}, \mathrm{U}^{-1}\underline{\hat{t_{n-1}}}, \mathrm{U}^{-1}\underline{\hat{t_n}}, \mathrm{U}^{-1}\underline{\hat{t_{n+1}}}\big) \Leftrightarrow \\ \underline{\hat{t_{n+2}}} = \mathrm{S_{II}}\big(\underline{\hat{t_1}}, \underline{\hat{t_{n-1}}}, \underline{\hat{t_n}}, \underline{\hat{t_{n+1}}}\big) \end{aligned} \tag{9.6.2}$$

for a public vector of dimension $2m$ with polynomial elements over $\mathbb{F}_q$. Analogously one now computes the disguised initial conditions $\mathrm{I} = \{\mathrm{U}\underline{t_0}, \mathrm{U}\underline{t_1}\}$ and the representations of the other components of $\mathcal{V}$; $\mathcal{W} := \{\mathrm{S}, \mathrm{S_I}, \mathrm{S_{II}}, \mathrm{S_{III}}, \mathrm{S_{IV}}\}$.

Corollary 9.6.1. *Under Definition 9.1.9 one now has a candidate BBG in the algebraic structure $Y = (\mathbb{A}^{2m}(\mathbb{F}_q), \{\mathcal{W}, \mathrm{I}\})$ which describes the XTR group operation on disguised trace elements.*

To compute with $Y = (\mathbb{A}^{2m}(\mathbb{F}_q), \{\mathcal{W}, \mathrm{I}\})$, a public/BBG–user acquires Y and computes $\underline{\hat{t_2}}$ from I and $\mathrm{R_I}$ given in Y. Using an $a \in \mathbb{Z}$ one can now iteratively evaluate $\underline{\hat{t_n}}$ using $\mathcal{W} \in Y$. Thus a user is able to compute the disguised group law of X without directly using the natural representation; specifically the basis $\mathcal{B}$ of $\mathbb{F}_q^{2m} \in X$.

9.6.2 Cryptanalysis of Disguised XTR

XTR fails in a similar manner to LUC did in §9.5.2 as one would expect given their structural similarity. This attack relies on the fact that a cryptanalyst does not need to use the matrices which describe the disguised group operation ($\mathcal{W} \in Y$) as they were intended. One in fact uses them to create a multiplication algorithm for the undisguised field $\mathbb{F}_{q^{2m}}$:

Lemma 9.6.2. *One can create a disguised field multiplication algorithm $\mathcal{A}$ for $\mathbb{F}_{q^{2m}}$ from the disguised group description given in Y.*

Proof. Set the multiplication algorithm $\mathcal{A}$ to be defined by

$$\mathcal{A}\big(\underline{\hat{x}}, \underline{\hat{y}}\big) := \mathrm{S_{II}}\big(\underline{\hat{x}}, \underline{0}, \underline{0}, \underline{\hat{y}}\big)$$

for two arbitrary $2m$–tuples $\underline{\hat{x}}$ and $\underline{\hat{y}}$. Then trivially, $\mathcal{A}$ is a disguised multiplication algorithm for $\mathbb{F}_{q^{2m}}$ since by equations (9.6.1) and (9.6.2) one has

$$\mathrm{U}^{-1}\mathrm{S_{II}}\big(\underline{\hat{t_1}}, \underline{\hat{t_{n-1}}}, \underline{\hat{t_n}}, \underline{\hat{t_{n+1}}}\big) = \underline{t_{n+2}} \sim \big(t_1 \cdot t_{n+1} - t_n \cdot (t_1)^{q^m} + t_{n-1}\big)$$

where $\mathrm{S_{II}} \in Y$ and hence $\mathrm{U}^{-1}\mathcal{A}\big(\underline{\hat{x}}, \underline{\hat{y}}\big) \sim xy \in \mathbb{F}_{q^{2m}}$. □

The attack now proceeds as follows:

Algorithm 9.6.1 (Attack of Disguised XTR)**.**

1. Eve acquires Y and picks a random $2m$–tuple $\underline{\hat{w}}$ which with a high degree of probability corresponds to some undisguised (and unknown to the attacker) $\underline{w}$. Clearly, one could pick $\underline{\hat{w}} = \underline{\hat{t_1}}$ which is known to correspond to the undisguised t_1.

2. Using Lemma 9.6.2 Eve constructs a disguised field multiplication algorithm $\mathcal{A}$ and uses this to recursively compute the set

   $$\Delta = \big\{\underline{\hat{w_1}}, \underline{\hat{w_2}}, \ldots, \underline{\hat{w_{2m+1}}}\big\} = \big\{\mathrm{U}\underline{w}, \mathrm{U}\underline{w^2}, \mathrm{U}\underline{w^3}, \ldots, \mathrm{U}\underline{w^{2m+1}}\big\}$$

   by iteratively evaluating $\underline{\hat{w_i}} = \mathcal{A}\big(\underline{\hat{w_{i-1}}}, \underline{\hat{w}}\big)$.

3. One now has $2m + 1$ vectors in an $2m$–dimensional vector space, so one can continue to find a defining polynomial for $\mathbb{F}_{q^{2m}}$ as we did in §9.2.2. With a high probability, one can now find a polynomial basis for $\mathbb{F}_{q^{2m}}$.

4. One now proceeds as before and computes a map between the disguised representation Y and our newly computed natural one X' as in §9.1.4.

9.7 Discussions & Remarks

It is hoped that disguising groups will enable us to define black–box groups suitable for cryptography. Such groups theoretically give us the optimum security for a given primitive and may lead to additional functionality.

However, as our work has shown generally disguising is a difficult thing to do. This is because a cryptanalyst can use the public group description in anyway he chooses, and not just to compute the disguised group law. As we presented, this usually allows an attacker to construct a multiplication formula of the underlying 'disguised' field. With this, he can construct a natural representation X' of Y equivalent to X. All that remains is for him to map the problem from $Y \to X'$ and he has undisguised the representation.

It was our hope that using a novel description of the torus T_2 and later LUC and XTR, that one could stop such a field multiplication oracle being formed. We showed that this is not the case. However, the practice of 'disguising' groups in a bid to construct BBGs has a more fundamental problem.

When attempting to create BBGs by disguising, one must perform additional computation to construct a system. Additionally, such systems then require

added bandwidth over a non–disguised primitive in order to describe the system. Hence, using disguising as a method to increase security seems fruitless: If so much work is required, both to construct and use such systems, why would this be done in place of just increasing the primitives security coefficient? Hence disguising general seems a poor way of trying to improve a primitives security.

This leads us to conclude that such systems are only useful if they give new cryptographic functionality (for example: a trapdoor pairing as was given in [19]) which is hard to get any other way.

Part III
Appendices

A
Detailed Methods

A.1 SEC 1 Parameter Representation

There are several different types of domain parameter which need to be represented in the initialisation of ECC systems, including finite field elements, natural numbers and points on elliptic curves. SEC 1 describes in detail how one should represent these data types for transmission and storage, which is done principally using octet strings and this is what we present in the sequel.

Methods for converting between one internal representation and another are trivial, however we do not require most of these in our analysis here and so we direct the interested reader to §2.3 of [15] for the details.

A.1.1 SEC 1: Bit–String–to–Octet–String conversion

Octet strings are created from bit–strings using the methods described in this section where by convention, we will be using the most–significant–bit (MSB) representation of binary strings.

Informally, one creates an octet string by padding a binary string with `0`'s on it left (MSB) hand–side to make its length a multiple of 8.

Definition A.1.1. [15] Formally, the conversion method from bit–strings to octet–strings is as follows:

INPUT: A bit string $\mathtt{B}$ of length b bits.

OUTPUT: An octet string $\mathtt{M}$ of length m octets.

Method: Convert the bit–string $\mathtt{B} = \mathtt{B}_0\mathtt{B}_1 \ldots \mathtt{B}_{b-1}$ to the octet–string $\mathtt{M} = \mathtt{M}_0\mathtt{M}_1 \ldots \mathtt{M}_{m-1}$ as follows:

1. **INPUT:** $\mathtt{B}$,
2. **for** $0 < i \leq m-1$ **set**
$$\mathtt{M}_i := \mathtt{B}_{b-8-8(m-1-i)}\mathtt{B}_{b-7-8(m-1-i)} \cdots \mathtt{B}_{b-1-8(m-1-i)},$$
3. Let $\mathtt{M}_0$ have its leftmost $8(m)-b$ bits set to `0`, and its rightmost $8-(8m-b)$ bits set to $\mathtt{B}_0\mathtt{B}_1 \ldots \mathtt{B}_{8-8m+b-1}$.
4. **OUTPUT:** $\mathtt{M}$.

One can deduce from Definition 5.2.1 that the bits required to transmit and store a general octet–string $\mathtt{M}$ here are $\|\mathtt{M}\| = 8\lceil b/8 \rceil$ where trivially $b = \|\mathtt{B}\|$.

A.1.2 SEC 1: Representing Integers

In SEC 1 [15] they describe a method to represent integers as octet–strings which we will present here. Informally the idea is to represent an integer in binary, and then convert the resulting bit–string to an octet–string as was described in Definition A.1.1.

Definition A.1.2. [15] Formally, the conversion method from integer elements to octet–strings is as follows:

INPUT: A non–negative integer $x \in \mathbb{N}$.

OUTPUT: An octet–string $\mathtt{M}$ of length m octets.

Method: Have a fixed length m of sufficient size, or compute the m required by noting that

$$2^{8m} > x$$

for a valid and unique representation to occur. It is trivial that $m \geq \lceil (\lg x)/8 \rceil$. Convert $x = x_{m-1}2^{8(m-1)} + x_{m-1}2^{8(m-2)} + \cdots + x_1 2^8 + x_0$ represented in base $256 = 2^8$ to an octet string as follows:

1. **INPUT:** An integer x,
2. **for** $0 \leq i \leq m-1$, **set**
$$\mathtt{M}_i := x_{m-1-i},$$
3. **OUTPUT:** $\mathtt{M}$.

Trivially, the bit–size required to represent these integers in this form is $\|x\| = 8m \geq 8\lceil (\lg x)/8 \rceil$.

A.1.3 SEC 1: Representing Finite Field elements

SEC 1 only considers how to represent elements of the two most common finite field types, namely prime fields $\mathbb{F}_p$ and binary fields $\mathbb{F}_{2^n}$. Here we closely follow the approach given in [15].

Informally, the idea is to consider the finite field element of $\mathbb{F}_p$ as an integer $x < p$. This integer is then converted into an octet–string using the method outlined in Definition A.1.2. With an element of $\mathbb{F}_{2^n}$, one views the coefficients of its polynomial representation as the bit–string, with the highest order coefficients ordered to the leftmost part of this string.

Definition A.1.3. [15] Formally, the conversion method from finite field elements to octet–strings is as follows:

INPUT: An element of the field $a \in \mathbb{F}_q$.

OUTPUT: An octet string $\mathtt{M}$ of length $m = \lceil (\lg q)/8 \rceil$ octets.

Method: Convert a to an octet string $\mathtt{M} = \mathtt{M}_0\mathtt{M}_1 \ldots \mathtt{M}_{m-1}$ as follows:

1. **INPUT:** $a \in \mathbb{F}_q$,
2. **if** $q = p$ is an odd prime, then a can be represented by an integer in the interval $[0, q-1]$. Convert a to $\mathtt{M}$ using the method described in Definition A.1.2.

3. **if** $q = 2^n$, then $a = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$ is a binary polynomial. Convert a to an octet string $\mathtt{M}$ as follows:

   (a) **for** $0 < i \leq m-1$, **set**:

   $$\mathtt{M}_i := a_{7+8(m-1-i)} a_{6+8(m-1-i)} \cdots a_{8(m-1-i)},$$

   (b) Let $\mathtt{M}_0$ have its leftmost $8m - n$ bits set to $\mathtt{0}$, and its rightmost $8 - (8m - n)$ bits set to $a_{m-1}a_{m-2}\ldots a_{8m-8}$.

4. **OUTPUT:** $\mathtt{M}$.

The bit–size required to represent the finite field element a is trivially $\|a\| = 8\lceil(\lg q)/8\rceil$ for both cases.

A.1.4 SEC 1: Representing Elliptic Curve Points

We present here the method outlined in SEC 1 for representing elliptic curve points. If we assume point–compression is being used, the idea is that the compressed ordinate (y–coordinate) is placed in the leftmost octet of the octet–string along with an indication that point–compression is 'on'. In this case, the abscissa (x–coordinate) is placed in the remainder of the octet–string. If point–compression is not being used, one places an indication of such in the leftmost octet followed by the remainder of the string being made up by the abscissa and ordinal.

Definition A.1.4. [15] Formally, the conversion method from elliptic curve points to octet–strings is as follows:

INPUT: A point $P \in E(\mathbb{F}_q)$ of the an elliptic curve represented affinely as (a, b).

OUTPUT: An octet–string $\mathtt{M}$ of length m octets where $m = 1$ if $P = \mathcal{O}$, $m = \lceil(\lg q)/8\rceil + 1$ if $P \neq \mathcal{O}$ and point–compression is being used or $m = 2\lceil(\lg q)/8\rceil + 1$ if $P \neq \mathcal{O}$ and point–compression is not being used.

Method: Convert P to an octet string $\mathtt{M} = \mathtt{M}_0\mathtt{M}_1\ldots\mathtt{M}_{m-1}$ as follows:

1. **INPUT:** $P = (x_p, y_p)$ on E.
2. **if** $P = \mathcal{O}$, **OUTPUT:** $\mathtt{M} = \mathtt{00}_{16}$.
3. **if** $P = (x_p, y_p) \neq \mathcal{O}$ and point–compression is being used:

   (a) Convert the finite field element x_p to an octet–string $\mathtt{X}$ of length $m = \lceil(\lg q)/8\rceil$ using Definition A.1.3.

   (b) Compute a single bit corresponding to which point is being represented, $\tilde{y_p}$, from y_p as follows:

      i. **if** $q = p$ is an odd prime, set $\tilde{y_p} = y_p \pmod 2$,

      ii. **else if** $q = 2^n$, set $\tilde{y_p} = 0$ if $x_p = 0$, otherwise compute $z = z_{m-1}x^{m-1} + \cdots + z_1 x + z_0$ such that $z = y_p x_p^{-1}$ and then set $\tilde{y_p} = z_0$.

   (c) Now, assign the value $\mathtt{02}_{16}$ to the single octet $\mathtt{Y}$ if $\tilde{y_p} = 0$ or the value $\mathtt{03}_{16}$ if $\tilde{y_p} = 1$.

(d) **OUTPUT:** $\mathtt{M} = \mathtt{Y} \| \mathtt{X}$.

4. **if** $P = (x_p, y_p) \neq \mathcal{O}$ and point–compression is not being used:

   (a) Convert the finite field element x_p to an octet–string $\mathtt{X}$ of length $m = \lceil (\lg q)/8 \rceil$ using Definition A.1.3.

   (b) Convert the finite field element y_p to an octet–string $\mathtt{Y}$ of length $m = \lceil (\lg q)/8 \rceil$ using Definition A.1.3.

   (c) **OUTPUT:** $\mathtt{M} = \mathtt{04}_{16} \| \mathtt{X} \| \mathtt{Y}$.

Clearly here, the leftmost octet $\mathtt{T}$ of $\mathtt{M}$ indicates whether $\mathtt{M}$ represents the point at infinity ($\mathtt{T} = \mathtt{00}_{16}$), or whether point–compression is being used ($\mathtt{T} = \mathtt{02}_{16}$ or $\mathtt{T} = \mathtt{03}_{16}$) or not ($\mathtt{T} = \mathtt{04}_{16}$).

The bit–sizes required to represent elliptic curve points is then

$$\|P\| = \begin{cases} 16 \lceil (\lg q)/8 \rceil + 8 & \text{without point–compression,} \\ 8 \lceil (\lg q)/8 \rceil + 8 & \text{with point–compression,} \\ 8 & \text{when } P = \mathcal{O}. \end{cases} \tag{A.1.1}$$

B Tables

B.1 Prime Tables

Type–1 Primes

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
4	(1,4,1)	4	(2,2,1)	5	(1,5,1)	6	(1,6,1)	6	(2,3,1)
8	(2,4,1)	8	(4,2,1)	9	(1,9,1)	9	(3,3,1)	10	(1,10,1)
10	(2,5,1)	10	(5,2,1)	11	(1,11,3)	12	(1,12,1)	12	(2,6,1)
12	(4,3,1)	14	(1,14,1)	14	(2,7,3)	16	(1,16,7)	16	(4,4,1)
17	(1,17,3)	18	(1,18,5)	18	(2,9,2)	18	(2,9,4)	18	(2,9,1)
18	(6,3,1)	18	(9,2,1)	20	(1,20,1)	20	(1,20,9)	20	(1,20,7)
20	(2,10,1)	20	(4,5,1)	20	(10,2,1)	21	(3,7,3)	21	(3,7,1)
21	(7,3,1)	22	(1,22,5)	22	(1,22,1)	22	(2,11,2)	22	(2,11,5)
24	(1,24,1)	24	(1,24,5)	24	(2,12,5)	24	(4,6,1)	26	(1,26,9)
26	(2,13,1)	29	(1,29,11)	29	(1,29,13)	29	(1,29,5)	29	(1,29,1)
30	(2,15,4)	30	(3,10,3)	32	(2,16,1)	32	(2,16,3)	32	(4,8,1)
33	(3,11,1)	33	(3,11,5)	36	(2,18,1)	36	(4,9,4)	36	(4,9,1)
36	(4,9,2)	36	(6,6,1)	36	(12,3,1)	36	(18,2,1)	38	(1,38,9)
38	(2,19,6)	38	(2,19,4)	40	(1,40,17)	40	(1,40,19)	40	(8,5,2)
42	(1,42,5)	42	(2,21,2)	42	(2,21,8)	42	(6,7,1)	44	(1,44,7)
44	(4,11,1)	44	(11,4,1)	45	(1,45,19)	45	(9,5,1)	46	(1,46,17)
46	(2,23,7)	48	(3,16,3)	48	(6,8,1)	48	(8,6,1)	48	(16,3,1)
49	(1,49,17)	51	(1,51,7)	52	(1,52,19)	54	(1,54,5)	54	(2,27,4)
56	(2,28,1)	56	(2,28,13)	56	(14,4,1)	57	(1,57,13)	57	(1,57,25)
59	(1,59,27)	60	(1,60,19)	60	(2,30,13)	60	(4,15,7)	60	(4,15,2)
60	(6,10,3)	61	(1,61,17)	61	(1,61,19)	62	(2,31,15)	64	(1,64,29)
64	(2,32,5)	64	(8,8,1)						

Table B.1: Type–1 Primes p with $4 \leq \lceil \lg p \rceil \leq 64$.

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
160	(1,160,31)	160	(4,40,19)	161	(1,161,11)	161	(23,7,1)	162	(2,81,16)
162	(2,81,26)	162	(2,81,32)	162	(6,27,4)	162	(6,27,10)	164	(2,82,27)
164	(4,41,19)	165	(3,55,13)	165	(5,33,5)	166	(2,83,7)	166	(2,83,1)
166	(2,83,19)	166	(2,83,31)	168	(1,168,19)	168	(1,168,79)	168	(2,84,41)
168	(3,56,9)	168	(3,56,11)	168	(6,28,3)	168	(6,28,11)	168	(8,21,1)
168	(14,12,5)	168	(42,4,1)	170	(34,5,1)	174	(1,174,77)	174	(1,174,1)
174	(2,87,2)	174	(2,87,17)	174	(2,87,43)	177	(1,177,53)	180	(1,180,11)
180	(4,45,11)	180	(5,36,11)	180	(5,36,7)	182	(2,91,37)	182	(14,13,1)
184	(1,184,19)	184	(1,184,5)	184	(46,4,1)	185	(1,185,67)	185	(1,185,49)
185	(1,185,71)	186	(2,93,22)	188	(2,94,25)	188	(4,47,6)	188	(4,47,15)
189	(1,189,53)	189	(1,189,71)	189	(7,27,7)	190	(5,38,1)	192	(1,192,67)
192	(1,192,55)	192	(2,96,11)	192	(3,64,11)	192	(16,12,1)	192	(64,3,1)
194	(1,194,5)	196	(2,98,11)	197	(1,197,37)	197	(1,197,31)	197	(1,197,61)

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
198	(1,198,73)	198	(2,99,2)	198	(22,9,4)	200	(1,200,21)	200	(2,100,11)
200	(2,100,47)	200	(20,10,1)	201	(1,201,17)	201	(3,67,21)	202	(1,202,77)
205	(1,205,29)	205	(1,205,83)	206	(2,103,11)	206	(2,103,46)	206	(2,103,12)
206	(2,103,1)	208	(1,208,77)	209	(1,209,5)	209	(1,209,71)	209	(1,209,47)
209	(11,19,7)	210	(6,35,1)	210	(14,15,4)	210	(14,15,7)	212	(1,212,59)
212	(1,212,89)	212	(1,212,31)	212	(2,106,37)	212	(2,106,21)	213	(1,213,91)
213	(1,213,1)	213	(3,71,31)	213	(3,71,3)	214	(1,214,41)	214	(1,214,49)
214	(1,214,97)	214	(1,214,89)	214	(2,107,8)	216	(36,6,1)	216	(108,2,1)
217	(1,217,83)	218	(1,218,57)	218	(1,218,5)	218	(2,109,24)	218	(2,109,4)
219	(3,73,29)	220	(1,220,19)	220	(1,220,103)	220	(10,22,1)	221	(1,221,1)
222	(1,222,109)	222	(2,111,28)	222	(2,111,41)	222	(2,111,46)	224	(1,224,23)
224	(2,112,5)	224	(4,56,9)	224	(7,32,5)				

Table B.2: Type–1 Primes p with $160 \leq \lceil \lg p \rceil \leq 224$.

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
224	(1,224,23)	224	(2,112,5)	224	(4,56,9)	224	(7,32,5)	225	(1,225,59)
225	(1,225,83)	225	(1,225,101)	225	(5,45,17)	226	(2,113,1)	226	(2,113,7)
227	(1,227,11)	228	(2,114,55)	229	(1,229,23)	229	(1,229,17)	230	(2,115,19)
232	(1,232,67)	232	(1,232,107)	232	(1,232,115)	232	(1,232,23)	233	(1,233,1)
234	(2,117,4)	234	(2,117,47)	234	(2,117,29)	234	(6,39,14)	234	(13,18,5)
235	(1,235,67)	236	(1,236,9)	237	(1,237,83)	237	(1,237,61)	237	(1,237,11)
237	(3,79,25)	237	(3,79,27)	238	(1,238,113)	238	(17,14,5)	240	(1,240,37)
240	(1,240,83)	240	(10,24,7)	240	(30,8,3)	242	(1,242,97)	242	(11,22,3)
243	(3,81,1)	244	(1,244,23)	245	(1,245,19)	245	(1,245,89)	246	(2,123,46)
246	(6,41,8)	248	(2,124,49)	248	(4,62,23)	248	(4,62,27)	248	(4,62,25)
248	(4,62,7)	248	(8,31,12)	249	(1,249,95)	250	(2,125,44)	250	(10,25,4)
250	(50,5,1)	251	(1,251,3)	252	(1,252,55)	252	(1,252,113)	252	(3,84,25)
252	(3,84,29)	252	(3,84,13)	252	(4,63,29)	252	(7,36,1)	252	(9,28,9)
252	(18,14,3)	254	(2,127,48)	254	(2,127,29)	256	(4,64,19)		

Table B.3: Type–1 Primes p with $224 \leq \lceil \lg p \rceil \leq 256$.

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
256	(4,64,19)	257	(1,257,73)	257	(1,257,31)	258	(1,258,37)	258	(2,129,19)
258	(2,129,20)	258	(2,129,10)	259	(1,259,115)	260	(1,260,79)	260	(2,130,61)
260	(26,10,1)	261	(1,261,19)	262	(1,262,13)	262	(1,262,113)	262	(2,131,55)
264	(1,264,131)	264	(2,132,37)	264	(4,66,17)	264	(6,44,13)	264	(11,24,1)
265	(5,53,23)	266	(1,266,1)	266	(2,133,43)	268	(1,268,79)	270	(1,270,73)
270	(2,135,7)	270	(2,135,61)	270	(15,18,7)	272	(1,272,105)	272	(1,272,73)
272	(2,136,37)	272	(8,34,5)	273	(1,273,59)	274	(1,274,125)	275	(1,275,7)
276	(1,276,91)	276	(6,46,13)	276	(12,23,6)	277	(1,277,121)	277	(1,277,101)
278	(1,278,117)	278	(1,278,29)	278	(1,278,25)	278	(2,139,15)	280	(1,280,131)
280	(7,40,11)	280	(8,35,11)	281	(1,281,103)	281	(1,281,13)	282	(3,94,27)
282	(3,94,23)	282	(3,94,15)	283	(1,283,139)	283	(1,283,103)	284	(1,284,127)
284	(1,284,73)	285	(3,95,7)	285	(3,95,41)	285	(3,95,1)	285	(3,95,17)
285	(5,57,1)	288	(1,288,139)	288	(36,8,1)	289	(1,289,25)	290	(1,290,97)
290	(1,290,41)	290	(2,145,28)	291	(1,291,31)	291	(3,97,13)	292	(2,146,35)
293	(1,293,39)	293	(1,293,75)	294	(1,294,101)	294	(1,294,37)	296	(1,296,19)
297	(3,99,47)	298	(2,149,61)	299	(1,299,83)	299	(1,299,99)	300	(1,300,137)
300	(2,150,17)	300	(4,75,19)	300	(15,20,9)	301	(1,301,101)	301	(1,301,143)
301	(1,301,47)	304	(1,304,71)	304	(1,304,107)	305	(5,61,3)	306	(1,306,89)
306	(2,153,28)	307	(1,307,131)	308	(2,154,5)	308	(2,154,57)	308	(14,22,9)
308	(22,14,1)	309	(1,309,43)	309	(3,103,27)	310	(2,155,38)	311	(1,311,39)
312	(1,312,71)	312	(2,156,11)	312	(8,39,7)	312	(12,26,11)	312	(12,26,7)
314	(1,314,65)	314	(1,314,85)	314	(2,157,78)	314	(2,157,12)	315	(1,315,139)
317	(1,317,145)	317	(1,317,5)	318	(2,159,40)	318	(2,159,44)	320	(1,320,129)
320	(4,80,37)	320	(5,64,21)	321	(1,321,91)	321	(3,107,1)	322	(1,322,121)
322	(1,322,101)	322	(2,161,47)	322	(7,46,7)	322	(161,2,1)	323	(1,323,55)

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
324	(1,324,23)	324	(2,162,77)	324	(6,54,5)	325	(25,13,5)	326	(2,163,76)
329	(1,329,89)	330	(2,165,19)	330	(2,165,62)	330	(5,66,29)	330	(10,33,8)
331	(1,331,43)	332	(1,332,71)	332	(1,332,145)	332	(1,332,163)	332	(2,166,65)
332	(2,166,53)	332	(4,83,9)	333	(3,111,5)	334	(2,167,80)	336	(1,336,41)
336	(1,336,1)	336	(4,84,1)	336	(6,56,11)	336	(8,42,17)	338	(2,169,57)
339	(1,339,79)	340	(1,340,11)	340	(4,85,19)	341	(11,31,7)	342	(2,171,16)
342	(6,57,1)	342	(18,19,4)	344	(1,344,65)	344	(2,172,25)	344	(2,172,69)
344	(2,172,13)	344	(2,172,73)	344	(8,43,15)	345	(1,345,137)	345	(5,69,31)
346	(1,346,49)	347	(1,347,151)	348	(4,87,2)	348	(6,58,17)	348	(6,58,23)
350	(1,350,9)	350	(1,350,173)	350	(1,350,29)	350	(2,175,53)	350	(2,175,12)
350	(5,70,13)	350	(14,25,11)	350	(25,14,5)	352	(1,352,103)	352	(1,352,115)
352	(1,352,167)	352	(1,352,71)	352	(4,88,35)	354	(2,177,47)	354	(2,177,82)
354	(2,177,17)	354	(6,59,6)	355	(1,355,167)	356	(1,356,161)	356	(1,356,169)
357	(3,119,25)	358	(1,358,113)	359	(1,359,15)	360	(1,360,151)	360	(2,180,31)
360	(2,180,73)	360	(8,45,13)	360	(9,40,19)	360	(10,36,11)	362	(1,362,149)
362	(2,181,66)	364	(7,52,5)	364	(7,52,23)	364	(14,26,7)	365	(1,365,149)
365	(5,73,31)	366	(2,183,47)	366	(6,61,22)	369	(3,123,7)	372	(2,186,65)
372	(3,124,27)	372	(4,93,34)	374	(2,187,3)	374	(2,187,19)	377	(1,377,157)
377	(1,377,71)	380	(2,190,3)	380	(2,190,27)	380	(2,190,17)	380	(10,38,3)
381	(1,381,143)	384	(1,384,185)	384	(1,384,125)	384	(6,64,31)	385	(1,385,169)
385	(7,55,13)	386	(1,386,81)	386	(1,386,89)	386	(2,193,89)	388	(4,97,25)
389	(1,389,119)	390	(2,195,43)	390	(2,195,32)	390	(10,39,11)	390	(30,13,5)
391	(1,391,167)	391	(1,391,151)	392	(2,196,87)	392	(14,28,13)	393	(1,393,95)
394	(1,394,121)	394	(1,394,109)	394	(1,394,185)	395	(1,395,139)	397	(1,397,91)
398	(1,398,77)	398	(1,398,89)	398	(1,398,189)	398	(2,199,64)	399	(3,133,37)
400	(1,400,41)	400	(10,40,17)	401	(1,401,31)	402	(1,402,193)	402	(1,402,13)
402	(2,201,26)	402	(3,134,35)	403	(1,403,179)	404	(2,202,99)	404	(2,202,57)
404	(4,101,2)	405	(1,405,167)	405	(1,405,23)	405	(1,405,83)	405	(15,27,5)
405	(45,9,1)	407	(1,407,71)	408	(1,408,109)	408	(1,408,103)	408	(3,136,39)
408	(8,51,16)	409	(1,409,89)	410	(1,410,133)	410	(2,205,77)	411	(1,411,91)
412	(1,412,115)	414	(1,414,89)	414	(1,414,121)	414	(2,207,97)	414	(2,207,2)
416	(2,208,49)	416	(4,104,37)	416	(8,52,7)	416	(208,2,1)	418	(1,418,49)
418	(2,209,56)	418	(38,11,5)	419	(1,419,203)	419	(1,419,27)	419	(1,419,39)
419	(1,419,103)	420	(3,140,59)	420	(4,105,11)	420	(4,105,34)	421	(1,421,11)
422	(1,422,197)	422	(1,422,41)	422	(1,422,205)	422	(2,211,75)	423	(3,141,13)
424	(1,424,29)	424	(1,424,19)	424	(1,424,131)	425	(1,425,107)	425	(1,425,77)
426	(2,213,58)	427	(1,427,115)	428	(2,214,3)	428	(2,214,41)	428	(4,107,6)
428	(4,107,9)	429	(1,429,29)	429	(1,429,41)	430	(1,430,161)	430	(5,86,29)
432	(1,432,203)	432	(3,144,5)	432	(3,144,71)	434	(1,434,41)	434	(2,217,83)
434	(2,217,108)	434	(7,62,27)	436	(2,218,41)	437	(1,437,167)	438	(1,438,109)
438	(1,438,149)	438	(2,219,10)	438	(2,219,26)	438	(2,219,41)	440	(1,440,43)
440	(1,440,109)	440	(1,440,47)	440	(5,88,1)	440	(20,22,3)	440	(55,8,1)
441	(1,441,53)	441	(1,441,169)	442	(1,442,205)	442	(2,221,95)	442	(2,221,97)
443	(1,443,35)	444	(1,444,155)	444	(3,148,21)	444	(4,111,1)	444	(6,74,35)
444	(6,74,33)	446	(1,446,209)	446	(1,446,181)	446	(2,223,94)	448	(2,224,5)
448	(224,2,1)	449	(1,449,149)	449	(1,449,167)	450	(10,45,16)	450	(18,25,12)
450	(225,2,1)	451	(1,451,47)	452	(1,452,1)	452	(2,226,15)	452	(4,113,15)
453	(1,453,143)	453	(1,453,71)	453	(1,453,101)	453	(3,151,61)	453	(3,151,37)
454	(2,227,37)	454	(2,227,19)	456	(2,228,67)	456	(6,76,23)	457	(1,457,101)
458	(1,458,113)	458	(1,458,105)	458	(2,229,46)	459	(9,51,19)	460	(1,460,227)
462	(3,154,3)	464	(1,464,79)	464	(1,464,101)	464	(2,232,23)	464	(2,232,85)
464	(8,58,13)	465	(1,465,43)	465	(5,93,41)	465	(15,31,13)	466	(1,466,101)
466	(2,233,97)	468	(4,117,1)	468	(4,117,23)	468	(12,39,8)	469	(1,469,137)
469	(1,469,37)	470	(1,470,181)	470	(2,235,48)	470	(2,235,78)	470	(10,47,18)
470	(10,47,1)	471	(1,471,31)	472	(1,472,215)	472	(1,472,187)	472	(1,472,127)
472	(2,236,71)	472	(2,236,95)	473	(1,473,139)	474	(2,237,74)	475	(1,475,7)
475	(1,475,11)	476	(4,119,48)	477	(1,477,131)	478	(2,239,109)	480	(2,240,103)
480	(3,160,43)	480	(5,96,37)	480	(24,20,7)	480	(240,2,1)	482	(1,482,41)
482	(1,482,25)	482	(2,241,55)	482	(2,241,101)	484	(4,121,31)	485	(1,485,47)
485	(5,97,3)	486	(2,243,94)	486	(2,243,86)	486	(2,243,74)	486	(2,243,31)
488	(4,122,1)	489	(1,489,175)	489	(1,489,241)	489	(1,489,83)	489	(3,163,49)

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
490	(1,490,89)	490	(2,245,122)	490	(2,245,109)	491	(1,491,59)	492	(1,492,7)
492	(3,164,45)	492	(6,82,21)	494	(1,494,205)	494	(2,247,88)	496	(8,62,1)
498	(2,249,121)	498	(2,249,8)	499	(1,499,95)	500	(4,125,51)	500	(10,50,13)
501	(1,501,19)	501	(3,167,27)	502	(2,251,125)	503	(1,503,159)	504	(1,504,145)
504	(2,252,83)	504	(3,168,61)	504	(21,24,11)	506	(2,253,61)	506	(2,253,21)
506	(2,253,109)	506	(2,253,36)	507	(1,507,103)	508	(1,508,113)	508	(4,127,55)
509	(1,509,211)	509	(1,509,53)	509	(1,509,241)	509	(1,509,221)	510	(1,510,77)
510	(2,255,58)	510	(3,170,27)	510	(3,170,79)	510	(30,17,2)	512	(1,512,127)
512	(2,256,95)	512	(32,16,1)	513	(1,513,191)	513	(1,513,35)	513	(1,513,101)
513	(19,27,7)	514	(1,514,149)	515	(1,515,239)	516	(3,172,17)	516	(4,129,29)
518	(1,518,97)	518	(2,259,23)	518	(2,259,108)	518	(14,37,6)	518	(14,37,13)
520	(1,520,17)	520	(2,260,79)	520	(2,260,61)	520	(10,52,19)	521	(1,521,253)
521	(1,521,181)	522	(2,261,77)	522	(2,261,53)	522	(2,261,11)	522	(2,261,20)
522	(6,87,19)	524	(1,524,145)	524	(2,262,85)	524	(2,262,63)	525	(1,525,19)
525	(1,525,73)	525	(3,175,9)	525	(15,35,11)	526	(1,526,149)	526	(2,263,13)
528	(2,264,103)	528	(6,88,1)	528	(16,33,5)	529	(1,529,101)	530	(1,530,29)
530	(1,530,113)	530	(2,265,41)	530	(2,265,108)	530	(2,265,83)	530	(2,265,131)
530	(2,265,12)	530	(10,53,8)	530	(53,10,1)	531	(3,177,17)	531	(9,59,3)
532	(2,266,125)	532	(2,266,23)	532	(7,76,23)	533	(1,533,29)	533	(1,533,243)
534	(2,267,112)	535	(5,107,43)	536	(2,268,31)	536	(2,268,79)	536	(4,134,51)
537	(1,537,211)	537	(1,537,215)	537	(3,179,27)	537	(3,179,1)	538	(2,269,128)
538	(2,269,85)	539	(1,539,103)	540	(2,270,19)	541	(1,541,197)	542	(2,271,130)
544	(1,544,251)	545	(1,545,1)	545	(5,109,25)	546	(2,273,58)	546	(7,78,23)
548	(1,548,79)	548	(1,548,105)	548	(1,548,239)	548	(2,274,103)	548	(4,137,66)
549	(1,549,53)	549	(1,549,73)	550	(1,550,233)	550	(2,275,1)	550	(2,275,53)
552	(1,552,275)	552	(1,552,235)	552	(4,138,61)	554	(1,554,193)	554	(1,554,117)
554	(1,554,269)	555	(3,185,9)	556	(1,556,107)	556	(1,556,179)	556	(1,556,19)
556	(4,139,46)	558	(18,31,11)	560	(1,560,173)	560	(1,560,151)	560	(4,140,57)
560	(7,80,27)	560	(16,35,12)	561	(1,561,137)	561	(1,561,59)	561	(1,561,251)
561	(1,561,37)	563	(1,563,59)	563	(1,563,3)	564	(1,564,121)	564	(1,564,25)
564	(1,564,143)	564	(3,188,51)	564	(3,188,53)	565	(1,565,151)	566	(1,566,257)
566	(1,566,101)	566	(2,283,136)	566	(2,283,27)	566	(2,283,24)	568	(1,568,125)
568	(8,71,29)	569	(1,569,219)	569	(1,569,157)	570	(1,570,73)	570	(2,285,41)
570	(2,285,16)	570	(6,95,8)	572	(1,572,129)	572	(1,572,31)	572	(4,143,36)
572	(143,4,1)								

Table B.4: Type–1 Primes p with $256 \leq \lceil \lg p \rceil \leq 572$.

Type–3 Primes

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
5	(1,5,3)	8	(1,8,5)	9	(1,9,5)	12	(1,12,7)	14	(2,7,4)
17	(1,17,11)	19	(1,19,11)	20	(1,20,11)	20	(4,5,3)	24	(1,24,13)
24	(3,8,5)	25	(1,25,13)	29	(1,29,17)	30	(2,15,8)	30	(6,5,3)
33	(1,33,19)	33	(3,11,7)	36	(1,36,19)	38	(2,19,10)	40	(1,40,23)
42	(2,21,11)	45	(1,45,29)	47	(1,47,31)	48	(1,48,29)	49	(1,49,31)
50	(2,25,13)	54	(2,27,14)	60	(3,20,11)	60	(3,20,13)	62	(1,62,41)

Table B.5: Type–3 Primes p with $4 \leq \lceil \lg p \rceil \leq 64$.

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
160	(1,160,91)	160	(2,80,43)	163	(1,163,103)	164	(4,41,21)	165	(1,165,101)
168	(3,56,29)	170	(1,170,97)	170	(2,85,54)	171	(3,57,37)	174	(3,58,31)
180	(1,180,119)	180	(5,36,19)	181	(1,181,107)	181	(1,181,103)	184	(1,184,107)
185	(1,185,113)	188	(1,188,103)	189	(1,189,113)	192	(1,192,127)	192	(3,64,35)
194	(2,97,58)	194	(2,97,63)	198	(2,99,59)	200	(1,200,109)	201	(1,201,101)
201	(3,67,37)	202	(2,101,56)	204	(12,17,9)	209	(1,209,135)	209	(1,209,127)
210	(2,105,61)	213	(1,213,131)	213	(1,213,115)	218	(1,218,117)	220	(1,220,113)
220	(4,55,31)	222	(1,222,145)	222	(6,37,21)	224	(1,224,149)	224	(2,112,59)
224	(2,112,65)								

Table B.6: Type–3 Primes p with $160 \leq \lceil \lg p \rceil \leq 224$.

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
224	(1,224,149)	224	(2,112,65)	224	(2,112,59)	226	(2,113,59)	227	(1,227,143)
227	(1,227,147)	228	(1,228,137)	230	(2,115,64)	230	(5,46,29)	234	(1,234,133)
235	(1,235,127)	236	(4,59,32)	238	(1,238,125)	241	(1,241,133)	244	(1,244,143)
248	(1,248,163)	248	(2,124,67)	248	(4,62,37)	250	(10,25,13)	252	(1,252,151)
252	(3,84,43)	252	(4,63,40)	253	(1,253,149)				

Table B.7: Type–3 Primes p with $224 \leq \lceil \lg p \rceil \leq 256$.

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
258	(2,129,65)	260	(2,130,73)	261	(1,261,131)	264	(1,264,133)	264	(1,264,173)
266	(14,19,12)	269	(1,269,169)	269	(1,269,163)	270	(2,135,79)	270	(3,90,47)
270	(54,5,3)	274	(1,274,149)	274	(2,137,73)	276	(4,69,37)	278	(2,139,71)
278	(2,139,82)	278	(2,139,75)	280	(2,140,71)	283	(1,283,143)	284	(2,142,93)
285	(3,95,49)	288	(2,144,85)	289	(1,289,191)	289	(1,289,167)	290	(1,290,177)
291	(3,97,49)	294	(2,147,82)	296	(2,148,83)	296	(4,74,43)	297	(1,297,151)
300	(1,300,163)	300	(3,100,51)	301	(1,301,151)	303	(3,101,53)	306	(2,153,98)
311	(1,311,183)	314	(2,157,90)	318	(2,159,95)	318	(2,159,80)	320	(1,320,169)
320	(1,320,201)	320	(4,80,53)	321	(1,321,193)	321	(1,321,203)	322	(2,161,97)
324	(1,324,193)	324	(1,324,175)	325	(1,325,197)	326	(2,163,84)	328	(8,41,23)
331	(1,331,199)	332	(4,83,51)	333	(1,333,193)	333	(1,333,181)	336	(1,336,211)
342	(2,171,86)	342	(2,171,103)	342	(2,171,101)	342	(9,38,21)	343	(1,343,191)
344	(4,86,55)	346	(2,173,104)	348	(4,87,55)	349	(1,349,211)	350	(2,175,88)
350	(2,175,113)	353	(1,353,199)	356	(4,89,54)	357	(1,357,235)	358	(1,358,209)
358	(2,179,115)	360	(1,360,203)	360	(20,18,11)	361	(1,361,223)	361	(19,19,11)
363	(1,363,199)	368	(4,92,47)	369	(1,369,227)	371	(1,371,227)	372	(1,372,191)
372	(1,372,199)	372	(1,372,239)	372	(3,124,75)	372	(4,93,59)	374	(2,187,103)
377	(1,377,193)	378	(2,189,122)	379	(1,379,239)	380	(1,380,199)	380	(4,95,56)
382	(2,191,97)	384	(2,192,127)	385	(1,385,247)	385	(1,385,211)	386	(2,193,124)
387	(3,129,85)	390	(2,195,121)	391	(1,391,239)	393	(3,131,73)	393	(3,131,67)
394	(2,197,107)	396	(2,198,107)	396	(2,198,113)	398	(2,199,102)	400	(16,25,16)
400	(16,25,13)	404	(1,404,223)	405	(1,405,233)	407	(1,407,263)	408	(2,204,109)
408	(4,102,67)	409	(1,409,241)	410	(1,410,213)	410	(2,205,129)	410	(5,82,45)
412	(1,412,227)	416	(1,416,259)	419	(1,419,243)	420	(2,210,131)	420	(12,35,23)
420	(15,28,15)	422	(2,211,107)	425	(1,425,251)	425	(1,425,229)	426	(1,426,229)
428	(2,214,127)	428	(4,107,69)	429	(1,429,281)	430	(10,43,25)	432	(2,216,125)
432	(54,8,5)	437	(1,437,281)	437	(1,437,277)	438	(2,219,119)	438	(2,219,139)
438	(3,146,75)	444	(4,111,61)	448	(14,32,19)	450	(2,225,137)	459	(1,459,295)
462	(2,231,125)	462	(2,231,148)	462	(2,231,128)	464	(1,464,247)	464	(1,464,293)
464	(8,58,33)	466	(2,233,137)	468	(3,156,101)	470	(1,470,261)	472	(1,472,311)
473	(1,473,291)	474	(3,158,91)	475	(1,475,299)	475	(19,25,13)	476	(1,476,271)
478	(2,239,122)	480	(6,80,43)	486	(1,486,277)	488	(2,244,131)	489	(3,163,97)
492	(6,82,47)	494	(2,247,137)	498	(3,166,99)	504	(9,56,37)	506	(2,253,144)
506	(22,23,12)	512	(1,512,269)	513	(3,171,113)	514	(2,257,143)	515	(1,515,279)
520	(1,520,317)	520	(2,260,139)	524	(2,262,169)	528	(1,528,299)	528	(2,264,163)
534	(2,267,163)	536	(1,536,301)	536	(4,134,89)	540	(6,90,47)	542	(1,542,345)
546	(26,21,11)	549	(1,549,311)	550	(1,550,349)	550	(22,25,13)	554	(1,554,321)

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
556	(4,139,82)	557	(1,557,355)	558	(1,558,289)	564	(2,282,175)	566	(2,283,163)
566	(2,283,159)	567	(27,21,13)	570	(6,95,48)	572	(1,572,329)		

Table B.8: Type–3 Primes p with $256 \leq \lceil \lg p \rceil \leq 572$.

Type–4 Primes

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
5	(1,5,2)	5	(1,5,1)	6	(2,3,1)	7	(1,7,1)	9	(1,9,2)
10	(2,5,2)	10	(2,5,1)	11	(1,11,5)	12	(2,6,1)	13	(1,13,5)
13	(1,13,1)	14	(2,7,2)	14	(2,7,1)	16	(4,4,1)	17	(1,17,8)
17	(1,17,6)	17	(1,17,5)	17	(1,17,1)	19	(1,19,9)	19	(1,19,5)
19	(1,19,1)	20	(2,10,1)	22	(2,11,1)	23	(1,23,4)	24	(2,12,1)
24	(6,4,1)	24	(8,3,1)	25	(1,25,12)	26	(2,13,6)	27	(1,27,11)
29	(1,29,6)	29	(1,29,9)	29	(1,29,2)	29	(1,29,8)	31	(1,31,9)
31	(1,31,1)	35	(5,7,1)	37	(1,37,5)	37	(1,37,17)	37	(1,37,18)
38	(2,19,5)	39	(3,13,1)	39	(3,13,4)	41	(1,41,5)	41	(1,41,6)
41	(1,41,10)	41	(1,41,16)	41	(1,41,12)	43	(1,43,8)	44	(2,22,7)
45	(9,5,1)	46	(2,23,3)	47	(1,47,7)	47	(1,47,20)	48	(2,24,7)
48	(4,12,5)	50	(2,25,7)	51	(3,17,3)	51	(17,3,1)	52	(4,13,3)
52	(4,13,5)	53	(1,53,9)	54	(2,27,4)	54	(6,9,4)	55	(1,55,9)
55	(1,55,7)	56	(4,14,5)	56	(8,7,3)	57	(3,19,6)	58	(2,29,3)
59	(1,59,11)	59	(1,59,28)	59	(1,59,12)	59	(1,59,13)	60	(2,30,7)
60	(6,10,3)	61	(1,61,5)	61	(1,61,1)	61	(1,61,26)	61	(1,61,24)
61	(1,61,21)	62	(2,31,8)	64	(2,32,5)	64	(4,16,3)	64	(8,8,3)

Table B.9: Type–4 Primes p with $4 \leq \lceil \lg p \rceil \leq 64$.

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
161	(1,161,60)	161	(1,161,72)	162	(2,81,28)	162	(6,27,13)	163	(1,163,61)
164	(2,82,13)	164	(2,82,3)	164	(2,82,39)	165	(1,165,38)	165	(3,55,3)
166	(2,83,27)	170	(2,85,4)	170	(10,17,3)	170	(34,5,1)	171	(3,57,17)
172	(2,86,35)	173	(1,173,33)	173	(1,173,14)	173	(1,173,53)	173	(1,173,81)
173	(1,173,9)	173	(1,173,25)	174	(2,87,1)	174	(2,87,10)	174	(2,87,40)
175	(1,175,67)	176	(2,88,37)	176	(4,44,5)	176	(16,11,5)	176	(16,11,3)
177	(1,177,68)	177	(3,59,19)	178	(2,89,42)	179	(1,179,29)	179	(1,179,85)
180	(4,45,8)	180	(6,30,13)	181	(1,181,77)	181	(1,181,14)	181	(1,181,22)
181	(1,181,9)	182	(2,91,15)	182	(2,91,45)	183	(3,61,19)	183	(3,61,7)
184	(2,92,9)	184	(2,92,21)	184	(2,92,27)	185	(1,185,14)	185	(1,185,28)
185	(1,185,58)	185	(5,37,10)	186	(6,31,2)	187	(1,187,24)	187	(1,187,9)
187	(1,187,16)	187	(11,17,8)	188	(4,47,5)	188	(4,47,4)	189	(9,21,8)
190	(2,95,39)	190	(2,95,7)	190	(10,19,3)	191	(1,191,31)	191	(1,191,24)
191	(1,191,64)	192	(2,96,13)	193	(1,193,5)	193	(1,193,30)	193	(1,193,41)
193	(1,193,18)	193	(1,193,82)	193	(1,193,50)	193	(1,193,21)	194	(2,97,32)
194	(2,97,36)	194	(2,97,9)	195	(1,195,17)	195	(1,195,89)	196	(4,49,1)
196	(4,49,3)	196	(28,7,2)	197	(1,197,64)	197	(1,197,80)	199	(1,199,89)
199	(1,199,25)	199	(1,199,83)	199	(1,199,55)	199	(1,199,11)	200	(4,50,3)
201	(3,67,12)	202	(2,101,38)	203	(1,203,81)	203	(7,29,7)	204	(68,3,1)
205	(5,41,17)	206	(2,103,17)	206	(2,103,15)	206	(2,103,48)	206	(2,103,3)
207	(3,69,25)	207	(9,23,9)	208	(8,26,3)	209	(1,209,98)	209	(1,209,90)
210	(2,105,52)	211	(1,211,12)	211	(1,211,61)	211	(1,211,24)	212	(4,53,17)
212	(4,53,8)	213	(1,213,2)	213	(3,71,19)	213	(3,71,6)	214	(2,107,53)
214	(2,107,4)	215	(1,215,59)	215	(1,215,23)	216	(72,3,1)	221	(1,221,42)
221	(1,221,2)	223	(1,223,61)	223	(1,223,67)	223	(1,223,31)	223	(1,223,105)
224	(2,112,3)	224	(4,56,5)	224	(32,7,3)				

Table B.10: Type–4 Primes p with $160 \leq \lceil \lg p \rceil \leq 224$.

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
223	(1,223,105)	223	(1,223,67)	223	(1,223,31)	223	(1,223,61)	224	(2,112,3)
224	(4,56,5)	224	(32,7,3)	225	(3,75,28)	226	(2,113,35)	226	(2,113,13)
227	(1,227,89)	229	(1,229,90)	229	(1,229,73)	230	(2,115,37)	232	(2,116,33)
233	(1,233,2)	233	(1,233,20)	233	(1,233,9)	234	(18,13,6)	235	(1,235,4)
235	(1,235,113)	235	(1,235,112)	235	(5,47,21)	236	(2,118,19)	236	(4,59,5)
237	(3,79,32)	238	(2,119,55)	239	(1,239,84)	239	(1,239,29)	239	(1,239,44)
241	(1,241,30)	241	(1,241,77)	241	(1,241,92)	242	(2,121,3)	243	(1,243,65)
243	(1,243,5)	245	(1,245,88)	246	(6,41,9)	246	(6,41,6)	246	(6,41,16)
247	(1,247,17)	248	(2,124,15)	250	(10,25,6)	250	(10,25,3)	251	(1,251,72)
251	(1,251,93)	252	(4,63,17)	252	(4,63,20)	254	(2,127,32)	254	(2,127,61)
254	(2,127,27)	255	(3,85,13)	255	(5,51,1)	255	(15,17,7)		

Table B.11: Type–4 Primes p with $224 \leq \lceil \lg p \rceil \leq 256$.

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
255	(3,85,13)	255	(5,51,1)	255	(15,17,7)	257	(1,257,45)	257	(1,257,60)
257	(1,257,108)	258	(6,43,10)	258	(6,43,18)	259	(1,259,124)	259	(1,259,76)
260	(2,130,27)	260	(10,26,5)	261	(3,87,22)	262	(2,131,44)	262	(2,131,59)
262	(2,131,28)	262	(2,131,46)	262	(2,131,61)	262	(2,131,34)	263	(1,263,45)
263	(1,263,28)	264	(6,44,15)	265	(1,265,109)	265	(5,53,18)	266	(2,133,1)
266	(2,133,17)	266	(14,19,9)	268	(2,134,63)	269	(1,269,53)	269	(1,269,13)
270	(6,45,14)	271	(1,271,120)	271	(1,271,116)	274	(2,137,8)	274	(2,137,25)
276	(6,46,15)	276	(6,46,7)	277	(1,277,102)	277	(1,277,24)	278	(2,139,14)
278	(2,139,29)	278	(2,139,32)	278	(2,139,12)	278	(2,139,58)	279	(1,279,89)
279	(3,93,16)	280	(4,70,31)	280	(20,14,3)	281	(1,281,81)	281	(1,281,140)
281	(1,281,122)	281	(1,281,49)	282	(2,141,10)	283	(1,283,91)	283	(1,283,48)
283	(1,283,28)	284	(2,142,43)	284	(2,142,9)	285	(15,19,6)	285	(15,19,3)
287	(1,287,20)	287	(1,287,65)	287	(1,287,121)	289	(1,289,114)	289	(1,289,33)
289	(1,289,116)	289	(1,289,12)	290	(2,145,57)	290	(2,145,12)	290	(2,145,41)
291	(1,291,17)	291	(3,97,27)	292	(2,146,39)	294	(2,147,19)	294	(2,147,64)
294	(6,49,11)	294	(6,49,3)	294	(42,7,1)	295	(1,295,9)	295	(5,59,23)
296	(4,74,15)	296	(8,37,10)	297	(9,33,13)	298	(2,149,44)	299	(1,299,128)
300	(50,6,1)	301	(1,301,30)	301	(1,301,97)	302	(2,151,15)	303	(1,303,17)
304	(2,152,59)	304	(4,76,7)	305	(1,305,28)	305	(1,305,137)	305	(5,61,30)
306	(6,51,25)	307	(1,307,8)	307	(1,307,19)	307	(1,307,57)	307	(1,307,132)
307	(1,307,76)	307	(1,307,115)	307	(1,307,9)	307	(1,307,124)	307	(1,307,136)
307	(1,307,17)	308	(2,154,43)	308	(28,11,5)	310	(2,155,4)	313	(1,313,74)
313	(1,313,140)	313	(1,313,97)	313	(1,313,73)	314	(2,157,14)	314	(2,157,7)
314	(2,157,59)	314	(2,157,46)	315	(3,105,8)	315	(105,3,1)	316	(2,158,53)
316	(4,79,5)	316	(4,79,36)	316	(4,79,29)	317	(1,317,149)	318	(2,159,43)
318	(2,159,49)	319	(1,319,115)	319	(11,29,5)	320	(4,80,23)	321	(1,321,86)
321	(1,321,20)	321	(3,107,42)	322	(2,161,71)	323	(1,323,13)	324	(2,162,61)
324	(2,162,13)	324	(6,54,25)	324	(12,27,5)	325	(5,65,6)	325	(13,25,8)
326	(2,163,16)	326	(2,163,19)	327	(1,327,131)	329	(1,329,138)	330	(2,165,4)
330	(2,165,19)	330	(6,55,4)	331	(1,331,43)	332	(2,166,43)	332	(2,166,9)
333	(1,333,158)	333	(3,111,55)	333	(9,37,18)	334	(2,167,53)	335	(1,335,9)
335	(5,67,25)	335	(5,67,13)	336	(2,168,1)	337	(1,337,149)	337	(1,337,18)
338	(2,169,24)	338	(2,169,2)	338	(2,169,55)	340	(4,85,22)	340	(10,34,13)
342	(2,171,64)	342	(6,57,8)	343	(1,343,131)	343	(1,343,72)	343	(1,343,51)
343	(1,343,117)	343	(7,49,4)	343	(7,49,17)	345	(5,69,28)	345	(15,23,10)
345	(23,15,4)	347	(1,347,60)	347	(1,347,164)	347	(1,347,16)	347	(1,347,120)
347	(1,347,101)	348	(4,87,5)	349	(1,349,133)	349	(1,349,54)	351	(1,351,149)
351	(1,351,23)	352	(2,176,69)	352	(2,176,35)	352	(2,176,21)	352	(8,44,15)
353	(1,353,124)	353	(1,353,101)	353	(1,353,73)	353	(1,353,168)	353	(1,353,125)
354	(6,59,19)	355	(1,355,169)	355	(5,71,17)	356	(4,89,21)	357	(1,357,122)
358	(2,179,28)	359	(1,359,36)	359	(1,359,113)	359	(1,359,160)	359	(1,359,96)
360	(6,60,17)	360	(12,30,1)	361	(1,361,137)	361	(1,361,21)	361	(1,361,101)
362	(2,181,58)	362	(2,181,84)	362	(2,181,18)	363	(1,363,59)	363	(1,363,107)
363	(3,121,49)	363	(3,121,43)	363	(3,121,16)	363	(3,121,36)	365	(1,365,168)

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
365	(5,73,6)	366	(2,183,10)	366	(2,183,34)	366	(6,61,28)	367	(1,367,121)
367	(1,367,32)	367	(1,367,181)	368	(4,92,45)	369	(1,369,38)	370	(2,185,89)
370	(2,185,51)	372	(62,6,1)	373	(1,373,98)	373	(1,373,48)	373	(1,373,33)
373	(1,373,102)	373	(1,373,109)	373	(1,373,94)	373	(1,373,114)	374	(2,187,53)
374	(2,187,92)	374	(2,187,27)	375	(3,125,13)	376	(4,94,37)	377	(1,377,9)
377	(1,377,122)	378	(18,21,10)	379	(1,379,53)	379	(1,379,101)	379	(1,379,112)
379	(1,379,188)	379	(1,379,61)	380	(2,190,51)	380	(4,95,3)	381	(1,381,182)
381	(3,127,50)	381	(3,127,31)	381	(3,127,10)	382	(2,191,76)	382	(2,191,71)
383	(1,383,13)	383	(1,383,33)	383	(1,383,5)	383	(1,383,68)	384	(12,32,5)
384	(16,24,5)	386	(2,193,84)	386	(2,193,24)	387	(9,43,16)	387	(9,43,5)
388	(2,194,5)	388	(2,194,3)	389	(1,389,26)	389	(1,389,81)	389	(1,389,136)
389	(1,389,146)	390	(30,13,5)	391	(1,391,177)	391	(1,391,19)	391	(1,391,7)
391	(1,391,67)	391	(1,391,80)	391	(23,17,5)	392	(4,98,3)	393	(1,393,164)
394	(2,197,54)	394	(2,197,74)	394	(2,197,44)	394	(2,197,94)	395	(1,395,19)
395	(5,79,28)	396	(2,198,91)	396	(4,99,35)	397	(1,397,29)	397	(1,397,97)
399	(3,133,37)	400	(2,200,27)	400	(2,200,17)	400	(80,5,2)	401	(1,401,5)
401	(1,401,137)	401	(1,401,86)	401	(1,401,84)	402	(2,201,100)	402	(2,201,64)
403	(1,403,188)	403	(1,403,28)	404	(2,202,75)	404	(4,101,3)	406	(58,7,3)
407	(1,407,91)	407	(1,407,21)	407	(1,407,115)	408	(2,204,19)	408	(2,204,25)
409	(1,409,45)	410	(10,41,19)	411	(3,137,67)	412	(2,206,69)	412	(2,206,45)
412	(2,206,99)	412	(2,206,83)	413	(1,413,73)	413	(1,413,88)	413	(7,59,27)
414	(6,69,14)	415	(1,415,143)	415	(5,83,19)	415	(5,83,4)	416	(2,208,25)
417	(3,139,55)	418	(22,19,9)	419	(1,419,45)	419	(1,419,204)	419	(1,419,139)
421	(1,421,165)	421	(1,421,57)	423	(1,423,119)	423	(141,3,1)	425	(1,425,42)
426	(6,71,21)	427	(1,427,11)	427	(1,427,40)	427	(1,427,125)	428	(2,214,103)
429	(3,143,71)	430	(10,43,11)	430	(10,43,17)	430	(10,43,1)	431	(1,431,200)
431	(1,431,155)	432	(6,72,35)	433	(1,433,129)	433	(1,433,168)	433	(1,433,97)
433	(1,433,73)	433	(1,433,102)	433	(1,433,72)	434	(2,217,24)	434	(2,217,86)
434	(2,217,15)	435	(3,145,11)	435	(15,29,13)	436	(2,218,53)	436	(2,218,63)
436	(4,109,31)	436	(4,109,21)	437	(1,437,40)	437	(1,437,197)	438	(2,219,79)
438	(6,73,35)	439	(1,439,200)	440	(8,55,14)	441	(21,21,10)	442	(2,221,79)
442	(26,17,1)	442	(26,17,5)	444	(2,222,31)	444	(4,111,17)	444	(4,111,50)
444	(12,37,10)	445	(1,445,69)	446	(2,223,103)	446	(2,223,25)	447	(1,447,161)
447	(3,149,5)	447	(3,149,13)	448	(4,112,55)	449	(1,449,21)	449	(1,449,13)
449	(1,449,29)	449	(1,449,116)	450	(2,225,16)	450	(10,45,14)	451	(11,41,12)
451	(41,11,3)	452	(2,226,1)	453	(3,151,8)	454	(2,227,64)	454	(2,227,40)
454	(2,227,69)	454	(2,227,94)	454	(2,227,3)	455	(1,455,193)	455	(13,35,4)
456	(4,114,29)	456	(6,76,27)	456	(12,38,3)	458	(2,229,53)	458	(2,229,83)
458	(2,229,85)	459	(9,51,11)	459	(51,9,1)	460	(20,23,1)	461	(1,461,221)
466	(2,233,83)	466	(2,233,22)	467	(1,467,187)	467	(1,467,208)	467	(1,467,9)
467	(1,467,65)	467	(1,467,91)	468	(2,234,115)	468	(4,117,5)	468	(4,117,20)
468	(4,117,17)	469	(1,469,65)	469	(1,469,72)	469	(1,469,128)	469	(1,469,216)
470	(10,47,17)	471	(1,471,11)	471	(1,471,65)	471	(3,157,60)	472	(2,236,33)
472	(2,236,21)	475	(1,475,64)	475	(1,475,67)	477	(1,477,218)	478	(2,239,104)
479	(1,479,52)	479	(1,479,181)	479	(1,479,129)	479	(1,479,116)	479	(1,479,204)
480	(20,24,7)	481	(1,481,213)	481	(1,481,165)	482	(2,241,85)	482	(2,241,59)
483	(3,161,57)	484	(4,121,19)	484	(4,121,49)	485	(1,485,62)	487	(1,487,25)
488	(2,244,115)	488	(8,61,30)	489	(3,163,28)	489	(3,163,26)	489	(3,163,11)
489	(3,163,23)	489	(3,163,15)	490	(10,49,9)	491	(1,491,120)	491	(1,491,44)
491	(1,491,84)	492	(6,82,27)	493	(1,493,33)	493	(1,493,240)	493	(1,493,49)
493	(1,493,214)	494	(2,247,112)	495	(1,495,107)	495	(5,99,1)	496	(2,248,111)
496	(62,8,3)	497	(1,497,108)	497	(7,71,14)	499	(1,499,99)	499	(1,499,64)
501	(1,501,110)	501	(3,167,47)	502	(2,251,45)	502	(2,251,123)	502	(2,251,85)
503	(1,503,107)	504	(24,21,8)	504	(168,3,1)	505	(1,505,109)	505	(1,505,222)
505	(5,101,6)	506	(46,11,5)	507	(3,169,57)	507	(3,169,28)	507	(3,169,75)
509	(1,509,146)	509	(1,509,162)	509	(1,509,73)	509	(1,509,160)	509	(1,509,246)
509	(1,509,248)	510	(10,51,23)	511	(1,511,145)	511	(1,511,81)	511	(7,73,32)
512	(32,16,1)	514	(2,257,128)	514	(2,257,57)	516	(86,6,1)	517	(1,517,114)
517	(1,517,41)	517	(1,517,216)	517	(1,517,18)	517	(1,517,230)	517	(11,47,7)
518	(2,259,108)	518	(14,37,5)	518	(14,37,6)	521	(1,521,112)	521	(1,521,1)
521	(1,521,97)	521	(1,521,114)	521	(1,521,153)	522	(2,261,85)	522	(2,261,16)
522	(6,87,28)	522	(6,87,32)	522	(18,29,7)	522	(18,29,10)	523	(1,523,91)

$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)	$\|p\|$	(w,d,c)
523	(1,523,184)	523	(1,523,181)	523	(1,523,244)	523	(1,523,53)	524	(2,262,21)
524	(2,262,81)	525	(3,175,48)	526	(2,263,103)	527	(1,527,88)	529	(1,529,168)
529	(1,529,5)	529	(1,529,89)	529	(1,529,188)	529	(1,529,262)	529	(1,529,242)
530	(2,265,46)	530	(2,265,31)	530	(2,265,7)	530	(10,53,4)	531	(1,531,65)
531	(3,177,68)	532	(4,133,60)	532	(28,19,6)	533	(1,533,122)	533	(1,533,160)
534	(6,89,22)	534	(6,89,6)	535	(1,535,157)	536	(2,268,117)	536	(4,134,5)
536	(8,67,7)	537	(3,179,82)	538	(2,269,63)	538	(2,269,84)	538	(2,269,69)
538	(2,269,8)	538	(2,269,54)	540	(4,135,8)	541	(1,541,270)	542	(2,271,111)
543	(3,181,51)	544	(8,68,23)	544	(32,17,3)	544	(32,17,1)	545	(1,545,108)
545	(1,545,268)	545	(1,545,154)	545	(1,545,2)	545	(1,545,248)	546	(2,273,76)
546	(2,273,10)	546	(14,39,19)	547	(1,547,185)	547	(1,547,248)	547	(1,547,120)
548	(4,137,26)	549	(3,183,82)	549	(3,183,11)	550	(2,275,47)	550	(2,275,21)
550	(10,55,2)	551	(1,551,35)	551	(1,551,92)	551	(1,551,131)	553	(1,553,209)
553	(1,553,205)	553	(1,553,101)	553	(1,553,18)	553	(1,553,12)	554	(2,277,97)
554	(2,277,69)	554	(2,277,65)	554	(2,277,106)	554	(2,277,10)	554	(2,277,90)
554	(2,277,51)	555	(15,37,3)	556	(2,278,27)	556	(4,139,23)	559	(1,559,236)
559	(1,559,189)	559	(1,559,163)	559	(1,559,164)	560	(2,280,129)	560	(112,5,1)
562	(2,281,94)	563	(1,563,64)	565	(1,565,152)	565	(1,565,9)	565	(1,565,42)
566	(2,283,11)	566	(2,283,14)	566	(2,283,75)	566	(2,283,10)	567	(21,27,7)
567	(63,9,4)	568	(2,284,5)	568	(4,142,21)	569	(1,569,64)	569	(1,569,53)
570	(6,95,39)	571	(1,571,97)	571	(1,571,96)	572	(2,286,133)		

Table B.12: Type–4 Primes p with $256 \leq \lceil \lg p \rceil \leq 572$.

Type–6 Primes

(w,c)	(w,c)	(w,c)	(w,c)	(w,c)	(w,c)	(w,c)	(w,c)	(w,c)
(3,3)	(4,1)	(4,3)	(5,5)	(6,3)	(7,3)	(8,1)	(8,7)	(9,9)
(10,7)	(10,9)	(11,5)	(12,3)	(15,3)	(15,11)	(15,15)	(16,1)	(16,3)
(16,7)	(16,15)	(18,3)	(18,7)	(18,9)	(20,7)	(20,13)	(21,17)	(22,15)
(23,9)	(23,11)	(23,15)	(26,15)	(28,3)	(28,7)	(29,11)	(30,3)	(30,7)
(30,9)	(30,15)	(30,19)	(31,11)	(32,15)	(33,17)	(33,29)	(34,25)	(36,31)
(37,9)	(37,29)	(38,7)	(38,13)	(39,23)	(39,39)	(40,15)	(40,27)	(41,27)
(42,15)	(43,29)	(43,39)	(44,7)	(44,21)	(44,27)	(46,15)	(47,5)	(47,9)
(47,41)	(48,21)	(51,21)	(52,21)	(52,37)	(53,5)	(53,41)	(55,3)	(55,11)
(55,51)	(57,9)	(57,35)	(60,33)	(61,15)	(61,21)	(61,57)	(63,29)	

Table B.13: Type–6 Primes p with $4 \leq \lceil \lg p \rceil \leq 64$.

(w,c)	(w,c)	(w,c)	(w,c)	(w,c)	(w,c)	(w,c)
(160,7)	(161,107)	(162,19)	(162,49)	(163,21)	(163,45)	(163,141)
(164,117)	(164,135)	(164,151)	(165,141)	(165,147)	(166,85)	(167,83)
(168,87)	(168,117)	(169,147)	(170,49)	(170,55)	(170,79)	(170,85)
(170,97)	(170,133)	(170,135)	(171,129)	(172,105)	(173,77)	(173,165)
(174,7)	(174,37)	(174,73)	(174,129)	(175,9)	(175,65)	(177,75)
(177,105)	(177,107)	(178,87)	(178,99)	(180,15)	(180,115)	(181,165)
(182,49)	(184,27)	(184,63)	(185,159)	(185,179)	(188,57)	(188,123)
(188,153)	(189,35)	(189,59)	(189,131)	(189,135)	(190,129)	(191,5)
(191,95)	(192,133)	(193,65)	(193,71)	(193,129)	(193,137)	(194,27)
(194,67)	(194,103)	(195,35)	(195,53)	(195,75)	(195,119)	(196,21)
(197,107)	(197,117)	(197,119)	(198,15)	(198,49)	(198,169)	(199,101)
(199,123)	(202,67)	(202,189)	(203,15)	(204,7)	(204,157)	(206,33)
(206,75)	(206,169)	(207,203)	(209,47)	(210,33)	(210,177)	(211,71)
(212,57)	(213,75)	(214,7)	(217,129)	(218,163)	(219,185)	(220,217)
(221,81)	(222,49)	(222,57)	(222,109)	(222,127)	(222,163)	(223,189)

Table B.14: Type–6 Primes p with $160 \leq \lceil \lg p \rceil \leq 224$.

(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)
(223,189)	(225,119)	(225,141)	(228,3)	(230,67)	(230,163)	(230,183)
(231,105)	(231,171)	(234,43)	(234,225)	(235,81)	(235,95)	(236,25)
(236,97)	(236,193)	(238,67)	(238,189)	(238,199)	(239,29)	(239,131)
(240,115)	(242,69)	(243,59)	(243,203)	(244,133)	(244,235)	(247,63)
(248,81)	(249,119)	(250,25)	(251,65)	(251,69)	(251,89)	(251,105)
(251,233)	(253,39)	(253,51)	(254,79)	(254,153)	(254,163)	(254,207)
(254,247)	(255,95)	(255,141)				

Table B.15: Type–6 Primes p with $224 \leq \lceil \lg p \rceil \leq 256$.

(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)
(255,95)	(255,141)	(257,155)	(258,73)	(260,223)	(261,105)	(261,137)
(262,79)	(263,9)	(263,35)	(263,83)	(264,175)	(265,77)	(265,137)
(266,133)	(266,135)	(266,193)	(268,43)	(268,111)	(268,121)	(269,245)
(270,127)	(270,133)	(272,57)	(272,81)	(272,163)	(272,223)	(273,5)
(273,35)	(274,63)	(274,67)	(274,177)	(276,157)	(277,101)	(277,179)
(277,267)	(278,117)	(279,89)	(280,45)	(281,197)	(282,159)	(282,189)
(285,17)	(286,43)	(287,35)	(287,65)	(287,273)	(288,127)	(289,101)
(291,71)	(292,13)	(292,91)	(293,239)	(294,67)	(294,169)	(294,205)
(294,249)	(295,9)	(295,143)	(295,291)	(296,61)	(296,163)	(296,247)
(297,11)	(297,17)	(297,215)	(297,227)	(297,245)	(297,249)	(300,157)
(301,27)	(301,165)	(301,261)	(303,101)	(304,37)	(305,131)	(305,285)
(306,45)	(306,73)	(306,117)	(307,135)	(308,27)	(308,81)	(310,15)
(310,189)	(310,265)	(310,283)	(311,111)	(311,303)	(312,91)	(313,125)
(313,279)	(313,305)	(314,93)	(314,277)	(314,307)	(315,81)	(315,203)
(315,219)	(316,87)	(317,9)	(317,317)	(319,9)	(319,101)	(319,123)
(319,281)	(319,293)	(320,27)	(320,261)	(321,165)	(322,219)	(322,297)
(323,273)	(325,137)	(325,315)	(326,249)	(327,9)	(327,233)	(328,15)
(329,39)	(329,239)	(330,93)	(330,295)	(331,203)	(333,285)	(335,173)
(335,183)	(335,239)	(336,241)	(337,71)	(337,105)	(337,177)	(338,97)
(338,99)	(339,185)	(339,275)	(339,303)	(340,291)	(341,5)	(341,77)
(342,15)	(342,85)	(343,225)	(343,269)	(344,231)	(344,343)	(345,107)
(348,241)	(351,143)	(351,345)	(352,55)	(352,81)	(352,223)	(353,141)
(353,329)	(353,335)	(355,141)	(357,231)	(357,347)	(359,23)	(359,141)
(360,105)	(360,313)	(361,281)	(363,309)	(365,239)	(366,309)	(367,69)
(368,127)	(368,265)	(369,65)	(370,69)	(370,349)	(371,329)	(372,147)
(374,289)	(375,81)	(375,171)	(376,115)	(376,301)	(376,331)	(378,25)
(379,99)	(380,127)	(382,255)	(383,369)	(384,231)	(384,331)	(385,189)
(386,235)	(386,267)	(386,357)	(388,133)	(389,89)	(390,3)	(390,133)
(392,207)	(392,351)	(393,77)	(393,101)	(393,249)	(393,291)	(393,327)
(393,371)	(394,97)	(394,169)	(394,267)	(395,29)	(396,231)	(396,253)
(396,261)	(397,387)	(398,127)	(399,51)	(400,181)	(403,293)	(404,55)
(404,63)	(404,265)	(406,129)	(406,169)	(406,405)	(407,309)	(408,37)
(408,231)	(408,247)	(408,267)	(408,325)	(408,385)	(409,29)	(409,125)
(409,381)	(409,399)	(410,33)	(410,375)	(411,63)	(411,333)	(411,345)
(412,67)	(412,115)	(412,385)	(413,221)	(413,317)	(413,411)	(414,315)
(415,323)	(416,235)	(417,245)	(418,163)	(419,299)	(420,45)	(421,105)
(421,261)	(422,163)	(422,273)	(423,309)	(424,163)	(424,423)	(425,57)
(425,329)	(425,387)	(426,423)	(427,69)	(427,183)	(428,345)	(429,131)
(430,73)	(431,173)	(433,335)	(434,103)	(436,295)	(436,303)	(436,337)
(437,117)	(437,249)	(437,425)	(438,25)	(438,213)	(439,101)	(440,187)
(441,227)	(441,257)	(442,189)	(443,53)	(444,267)	(445,41)	(445,95)
(445,227)	(445,269)	(446,33)	(446,435)	(447,99)	(448,211)	(450,213)
(450,225)	(450,289)	(451,23)	(451,165)	(452,37)	(452,261)	(452,301)
(453,89)	(453,165)	(453,221)	(454,375)	(454,433)	(455,105)	(455,393)
(456,21)	(456,243)	(456,375)	(458,127)	(459,101)	(459,285)	(459,371)
(461,321)	(461,369)	(463,81)	(465,29)	(466,423)	(467,329)	(470,99)
(470,375)	(472,445)	(473,149)	(473,327)	(474,309)	(476,223)	(476,397)
(477,315)	(479,221)	(480,165)	(480,345)	(481,65)	(481,237)	(482,475)
(483,329)	(484,333)	(485,165)	(486,207)	(486,345)	(487,443)	(489,179)
(489,345)	(492,21)	(492,247)	(494,375)	(495,51)	(495,443)	(497,159)

(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)
(497,245)	(498,375)	(498,409)	(499,161)	(500,55)	(500,135)	(500,321)
(502,49)	(502,135)	(502,339)	(502,343)	(503,165)	(503,309)	(504,133)
(505,51)	(505,179)	(505,227)	(505,377)	(505,461)	(507,351)	(507,359)
(508,15)	(509,35)	(509,167)	(509,179)	(509,207)	(509,281)	(510,15)
(510,37)	(510,325)	(511,111)	(512,75)	(512,145)	(512,285)	(513,159)
(513,261)	(513,431)	(514,169)	(514,255)	(515,15)	(516,121)	(516,303)
(517,357)	(517,405)	(518,37)	(518,57)	(518,99)	(519,393)	(520,513)
(522,345)	(523,375)	(524,117)	(524,427)	(527,249)	(527,509)	(528,381)
(529,69)	(530,189)	(531,375)	(531,411)	(532,123)	(532,331)	(533,279)
(535,83)	(536,37)	(536,261)	(536,403)	(538,87)	(538,213)	(538,345)
(538,465)	(538,499)	(539,119)	(539,243)	(539,393)	(540,31)	(540,91)
(540,183)	(540,421)	(540,457)	(542,93)	(542,175)	(542,439)	(544,163)
(544,477)	(546,15)	(546,393)	(546,399)	(548,453)	(549,261)	(549,435)
(549,537)	(551,335)	(552,81)	(552,487)	(553,549)	(554,417)	(555,345)
(556,63)	(557,357)	(557,377)	(557,467)	(557,531)	(558,33)	(558,39)
(558,99)	(559,153)	(559,231)	(559,273)	(559,503)	(559,519)	(560,211)
(562,45)	(562,147)	(562,153)	(562,415)	(563,459)	(564,115)	(564,253)
(564,351)	(566,427)	(566,445)	(568,51)	(569,369)	(570,25)	(571,71)
(571,183)	(571,221)	(571,261)	(572,57)	(572,525)	(572,555)	

Table B.16: Type–6 Primes p with $256 \leq \lceil \lg p \rceil \leq 572$.

Type–8 Primes

(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)
(4,3)	(5,1)	(5,3)	(6,3)	(6,5)	(7,1)	(8,5)	(9,3)
(9,9)	(10,3)	(10,5)	(11,9)	(12,3)	(12,5)	(13,1)	(13,13)
(14,3)	(16,15)	(17,1)	(17,9)	(17,13)	(18,5)	(18,11)	(18,17)
(19,1)	(19,19)	(20,3)	(20,5)	(20,17)	(21,9)	(21,19)	(21,21)
(22,3)	(22,17)	(23,15)	(23,21)	(24,3)	(24,17)	(26,5)	(29,3)
(31,1)	(31,19)	(32,5)	(32,17)	(33,9)	(33,25)	(35,31)	(36,5)
(36,17)	(36,23)	(37,25)	(37,31)	(39,7)	(39,19)	(41,21)	(41,31)
(42,11)	(42,17)	(42,33)	(44,17)	(46,21)	(50,27)	(50,35)	(52,47)
(54,33)	(54,53)	(55,55)	(56,5)	(56,27)	(56,47)	(57,13)	(57,25)
(57,49)	(58,27)	(58,57)	(59,55)	(61,1)	(61,31)	(61,45)	(62,57)
(63,25)	(64,59)						

Table B.17: Type–8 Primes p with $4 \leq \lceil \lg p \rceil \leq 64$.

(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)
(160,47)	(160,57)	(160,75)	(161,159)	(162,101)	(163,55)	(163,69)
(164,63)	(164,155)	(165,25)	(165,61)	(165,115)	(166,5)	(167,135)
(170,143)	(170,153)	(171,19)	(172,95)	(172,155)	(173,55)	(173,103)
(174,3)	(174,17)	(174,143)	(174,153)	(174,161)	(178,41)	(179,49)
(179,159)	(180,47)	(180,107)	(181,165)	(182,161)	(183,147)	(184,33)
(184,59)	(187,85)	(187,105)	(188,125)	(188,143)	(188,167)	(188,173)
(189,25)	(190,11)	(190,33)	(190,95)	(191,19)	(191,51)	(191,69)
(191,139)	(193,31)	(193,123)	(194,33)	(194,75)	(195,135)	(196,15)
(196,47)	(197,75)	(197,111)	(197,169)	(198,17)	(198,45)	(199,49)
(199,189)	(200,75)	(200,117)	(201,55)	(201,111)	(202,183)	(202,195)
(203,159)	(203,187)	(204,167)	(205,81)	(206,5)	(206,63)	(207,91)
(207,157)	(209,33)	(210,47)	(210,65)	(210,165)	(210,171)	(210,203)
(211,175)	(212,23)	(212,29)	(212,99)	(212,149)	(213,3)	(213,61)
(213,123)	(214,185)	(215,157)	(217,61)	(218,33)	(218,117)	(219,121)
(220,77)	(220,167)	(221,3)	(221,133)	(222,117)	(224,63)	

Table B.18: Type–8 Primes p with $160 \leq \lceil \lg p \rceil \leq 224$.

(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)
(224,63)	(225,49)	(225,81)	(225,103)	(226,5)	(226,77)	(226,155)
(226,203)	(226,215)	(228,93)	(228,149)	(228,185)	(229,91)	(230,27)
(230,77)	(231,165)	(231,217)	(233,3)	(233,159)	(234,83)	(235,15)
(235,151)	(235,181)	(236,209)	(237,181)	(238,161)	(238,215)	(239,87)
(239,199)	(241,39)	(242,63)	(243,9)	(243,31)	(243,199)	(244,189)
(245,163)	(246,107)	(246,171)	(246,243)	(247,81)	(248,237)	(249,75)
(250,207)	(251,9)	(252,129)	(252,143)	(254,245)	(255,19)	(255,31)
(256,189)						

Table B.19: Type–8 Primes p with $224 \leq \lceil \lg p \rceil \leq 256$.

(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)
(255,19)	(255,31)	(256,189)	(257,93)	(258,87)	(260,149)	(261,223)
(261,261)	(262,71)	(265,49)	(265,115)	(265,139)	(265,211)	(266,3)
(266,213)	(267,265)	(268,77)	(269,241)	(270,53)	(271,169)	(272,237)
(273,205)	(275,129)	(275,199)	(275,205)	(276,89)	(277,103)	(277,181)
(278,93)	(279,69)	(279,231)	(280,47)	(280,105)	(280,195)	(281,139)
(281,259)	(282,83)	(282,93)	(282,237)	(282,245)	(283,45)	(284,173)
(285,9)	(285,33)	(286,165)	(287,115)	(288,167)	(290,47)	(290,83)
(291,19)	(292,167)	(292,197)	(292,207)	(294,35)	(294,135)	(294,177)
(294,245)	(295,171)	(296,285)	(297,123)	(297,171)	(297,285)	(299,69)
(300,153)	(300,185)	(301,265)	(302,267)	(302,293)	(303,121)	(303,207)
(303,241)	(303,249)	(304,75)	(305,103)	(307,99)	(307,147)	(307,187)
(307,255)	(308,159)	(308,189)	(310,77)	(310,255)	(311,45)	(311,75)
(311,181)	(311,199)	(311,271)	(311,297)	(312,203)	(313,139)	(314,113)
(316,57)	(316,243)	(316,267)	(316,293)	(317,33)	(318,165)	(318,275)
(320,197)	(321,9)	(321,45)	(322,11)	(322,185)	(323,141)	(323,247)
(323,309)	(324,23)	(324,177)	(326,101)	(326,117)	(326,215)	(328,155)
(329,139)	(329,243)	(329,273)	(330,255)	(331,61)	(334,243)	(335,321)
(336,3)	(336,17)	(336,303)	(337,75)	(338,15)	(338,275)	(338,315)
(339,147)	(340,293)	(340,299)	(341,229)	(342,65)	(342,237)	(342,263)
(342,317)	(343,199)	(343,291)	(344,119)	(344,219)	(346,45)	(346,57)
(346,197)	(347,211)	(348,117)	(348,143)	(348,257)	(349,285)	(350,113)
(350,131)	(351,61)	(351,135)	(351,291)	(353,139)	(354,153)	(354,251)
(355,49)	(356,173)	(356,227)	(357,243)	(363,75)	(363,97)	(363,105)
(363,111)	(365,169)	(365,261)	(365,295)	(365,325)	(366,167)	(368,315)
(369,25)	(369,195)	(372,177)	(372,207)	(373,333)	(374,65)	(374,153)
(374,371)	(376,57)	(377,259)	(377,321)	(379,19)	(379,99)	(379,319)
(379,355)	(380,65)	(381,313)	(382,105)	(382,227)	(383,31)	(383,187)
(383,367)	(384,317)	(385,265)	(386,231)	(388,45)	(388,63)	(388,269)
(388,347)	(389,21)	(389,51)	(389,103)	(389,313)	(390,137)	(390,293)
(390,383)	(391,105)	(391,127)	(392,107)	(392,299)	(393,93)	(393,331)
(394,377)	(397,81)	(398,131)	(398,231)	(399,91)	(399,219)	(401,31)
(401,261)	(402,53)	(403,379)	(404,257)	(404,309)	(404,395)	(405,235)
(405,331)	(407,157)	(407,171)	(407,225)	(407,321)	(409,103)	(410,51)
(411,205)	(411,339)	(411,355)	(412,183)	(412,243)	(412,335)	(413,21)
(414,17)	(414,35)	(415,45)	(415,55)	(415,267)	(415,361)	(415,391)
(418,147)	(418,297)	(419,69)	(419,219)	(420,317)	(420,335)	(421,271)
(422,101)	(422,233)	(422,377)	(423,91)	(424,389)	(424,417)	(425,301)
(425,379)	(425,391)	(426,321)	(428,65)	(428,387)	(428,413)	(429,55)
(431,201)	(431,225)	(431,325)	(432,299)	(432,335)	(434,183)	(434,201)
(434,287)	(434,323)	(435,97)	(435,349)	(435,387)	(436,87)	(436,117)
(436,249)	(437,93)	(438,417)	(439,151)	(439,169)	(440,33)	(440,173)
(441,361)	(444,17)	(444,173)	(445,195)	(446,77)	(446,393)	(447,325)
(447,441)	(448,203)	(448,207)	(449,241)	(449,373)	(452,3)	(452,183)
(453,285)	(454,57)	(454,63)	(454,155)	(455,217)	(457,273)	(457,349)
(458,57)	(458,185)	(460,77)	(460,113)	(460,147)	(462,195)	(463,157)
(463,405)	(463,441)	(464,437)	(468,17)	(468,167)	(469,283)	(470,35)
(470,51)	(470,215)	(471,147)	(472,209)	(474,135)	(475,129)	(475,379)
(476,153)	(476,155)	(476,315)	(477,123)	(477,313)	(478,95)	(478,341)
(480,47)	(480,423)	(481,273)	(481,349)	(481,411)	(482,275)	(483,301)

(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)	(w, c)
(483,345)	(483,381)	(484,39)	(484,117)	(484,389)	(484,465)	(486,143)
(487,57)	(487,339)	(488,17)	(488,345)	(489,21)	(489,115)	(489,195)
(489,279)	(489,345)	(489,375)	(489,403)	(489,405)	(491,75)	(491,97)
(491,369)	(492,129)	(492,173)	(492,189)	(493,433)	(493,439)	(493,469)
(494,125)	(495,31)	(496,257)	(499,151)	(501,45)	(501,91)	(503,91)
(504,503)	(505,91)	(505,103)	(505,139)	(506,45)	(506,243)	(506,455)
(507,265)	(507,417)	(508,243)	(510,75)	(510,357)	(510,443)	(511,187)
(511,339)	(511,481)	(513,445)	(515,225)	(517,489)	(517,493)	(519,91)
(519,387)	(520,383)	(521,1)	(521,115)	(521,271)	(521,489)	(522,117)
(524,465)	(525,363)	(526,363)	(526,497)	(528,65)	(528,189)	(528,303)
(529,31)	(529,309)	(531,315)	(531,525)	(532,335)	(533,153)	(533,159)
(533,453)	(534,521)	(535,367)	(535,405)	(535,447)	(536,149)	(537,9)
(538,53)	(538,71)	(540,335)	(541,81)	(541,169)	(541,339)	(543,157)
(543,427)	(543,487)	(545,3)	(546,11)	(546,227)	(547,537)	(550,5)
(550,77)	(550,227)	(550,333)	(551,231)	(552,503)	(553,441)	(557,339)
(557,483)	(558,261)	(558,485)	(559,411)	(561,153)	(563,9)	(563,55)
(563,87)	(563,439)	(564,95)	(564,345)	(564,455)	(565,45)	(565,403)
(565,475)	(565,511)	(566,113)	(566,243)	(567,297)	(569,199)	(569,535)
(570,261)	(570,453)	(570,555)	(571,369)	(572,275)		

Table B.20: Type–8 Primes p with $256 \leq \lceil \lg p \rceil \leq 572$.

B.2 Miscellaneous Tables

163	0.6601618000305602	229	0.6601618000002933
293	0.6601618000000000	373	0.6601618000000010
443	0.6601626400287576	167	0.6601618000001196
233	0.6601621375781488	307	0.6601618000000032
379	0.6601618000000000	449	0.6601618000004183
173	0.6601618000015275	239	0.6601648838772995
311	0.6601618000000231	383	0.6601618000003180
457	0.6601618000000000	179	0.6601672583567503
241	0.6601618000000014	313	0.6601618000000055
389	0.6601618000000002	461	0.6601618862559711
181	0.6601618003503314	251	0.6601644146853504
317	0.6601618072986696	397	0.6601619331669182
463	0.6601618053460310	191	0.6601663122607721
257	0.6601618000000000	331	0.6601618000000000
401	0.6601618000000000	467	0.6601618000000000
193	0.6601618000000035	263	0.6601618011782448
337	0.6601618019934138	409	0.6601618000000000
479	0.6601618000000006	197	0.6601618117785750
269	0.6601618000000035	347	0.6601618000000000
419	0.6601627389568755	487	0.6601618278293675
199	0.6601618000000000	271	0.6601618000000000
349	0.6601618000000000	421	0.6601618000000000
491	0.6601624838900602	211	0.6601618028601703
277	0.6601618000005250	353	0.6601618000007601
431	0.6601627429412305	499	0.6601618015028997
223	0.6601618019916406	281	0.6601618001007970
359	0.6601630787894492	433	0.6601618000000000
503	0.6601618000000000	227	0.6601618000000000
283	0.6601618071297543	367	0.6601618042396075
439	0.6601618000000001	509	0.6601618000000041

Table B.21: Values of c_q for $q = 2^n$, n Prime.

Bibliography

[1] B. Allombert, *Explicit Computation of Isomorphisms between Finite Fields*, Finite Fields and their Applications **8** (2002), no. 3, 332–342.

[2] ANSI, *ANSI X9.63-2001: Public Key Cryptography for the Financial Services Industry, Key Agreement and Key Transport Using Elliptic Curve Cryptography*, American National Standards, 2001.

[3] K. Aoki, J. Franke, T. Kleinjung, A. K. Lenstra, and D. A. Osvik, *A kilobit special number field sieve factorization*, Advances in Cryptology - ASIACRYPT 2007: 13th International Conference on the Theory and Application of Cryptology and Information Security, Lecture Notes in Computer Science, vol. 4833, Springer–Verlag, Berlin, November 2007, pp. 1–12.

[4] ———, *A kilobit special number field sieve factorization*, Tech. report, Number Theory Archive, May 2007.

[5] D. Bailey, *Optimal Extension Fields for Fast Arithmetic in Public–Key Algorithms*, Advances in Cryptology - CRYPTO '98: 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings (H. Krawczyk, ed.), Lecture Notes in Computer Science, no. 1462, Springer–Verlag, New York, 1998, pp. 472 – 485.

[6] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, *Recommendation for Key Managment – Part 1: General (Revised)*, National Institute of Standards and Technology (NIST), March 2007.

[7] D. J. Bernstein and T. Lange, *Faster Addition and Doubling on Elliptic Curves*, Advances in Cryptology - ASIACRYPT 2007: 13th International Conference on the Theory and Application of Cryptology and Information Security, Lecture Notes in Computer Science, vol. 4833, Springer–Verlag, Berlin, November 2007, pp. 29–50.

[8] ———, *Inverted Edwards coordinates*, 17th Applied Algebra, Algebraic Algorithms and Error Correcting Codes Symposium (AAECC 2007), Lecture Notes in Computer Science, vol. 4851, Springer–Verlag, Berlin, November 2007, pp. 20–27.

[9] I. F. Blake, G. Seroussi, and N. P. Smart, *Elliptic Curves in Cryptography*, Second ed., London Mathematical Society, Lecture Note Series, no. 265, Cambridge University Press, 2002.

[10] D. Boneh and R. Lipton, *Searching for elements in black box fields and applications*, Lecture Notes in Computer Science - CRYPTO '96, vol. 1109, 1996.

[11] E. Brown, B.T. Myers, and J.A. Solinas, *Elliptic Curves with Compact Parameters*, CORR 2001-68, Technical Report, Centre for Applied Cryptographic Research, University of waterloo, Canada (2001), 1 – 17.

[12] ———, *Hyperelliptic Curves with Compact Parameters*, Designs, Codes and Cryptography **36** (2005), 245 – 261.

[13] Certicom Research, Certicom Corporation, *SEC 1: Elliptic Curve Cryptography*, SECG: Standards For Efficient Cryptography Group, 2000.

[14] ———, *SEC 2: Recommended Elliptic Curve Domain Parameters*, SECG: Standards For Efficient Cryptography Group, 2000.

[15] ———, *SEC 1: Elliptic Curve Cryptography (Working Draft Revision 1.7)*, SECG: Standards For Efficient Cryptography Group, November 2006.

[16] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, First ed., Discrete Mathematics and its Applications, Chapman & Hall / CRC Press, 2005.

[17] D. Coppersmith, *Modifications to the Number Field Sieve*, Journal of Cryptology (1993), no. 6, 169 – 180.

[18] R. Crandall and C. B. Pomerance, *Prime Numbers: A Computational Perspective*, 2nd ed., Springer–Verlag New York Inc, 2005.

[19] A. W. Dent and S. D. Galbraith, *Hidden pairings and trapdoor DDH groups*, Lecture Notes in Computer Science, vol. 4076, Springer–Verlag, Berlin, October 2006, pp. 436–451.

[20] W. Diffie and M.E. Hellman, *New Directions in Cryptography*, IEEE Transactions Information Theory **IT-22** (1976), no. 6, 644 – 654.

[21] H. M. Edwards, *A Normal Form for Elliptic Curves*, Bulletin of the American Mathematical Society **44** (2007), 393–422.

[22] T. ElGamal, *A Public–Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Transactions on Information Theory, vol. 31, 1985, pp. 469–472.

[23] A. Enge, *Elliptic Curves and Their Applications to Cryptography: An Introduction*, Kluwer Academic Publishers, 1999.

[24] J. C. Faugère and L. Perret, *Polynomial equivalence problems: Algorithmic and theoretical aspects*, EUROCRYPT 2006 (2006), 1 – 18.

[25] N. Ferguson and B. Schneier, *Practical Cryptography*, First ed., Wiley, 2003.

[26] G. Frey, *How to disguise an elliptic curve (weil decent)*, Talk at ECC '98, September 1998.

[27] G. Frey and H.–G. Rück, *A remark regarding m–divisibility and the discrete logarithm problem in the divisor class group of curves*, Mathematics of Computation **62** (1994), 865–874.

[28] S. D. Galbraith, *Disguising tori and elliptic curves*, preprint, 2006.

[29] S. D. Galbraith, F. Heß, and N. P. Smart, *Extending the GHS Weil descent attack*, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands (L. Knudsen, ed.), Lecture Notes in Computer Science, no. 2332, Springer–Verlag, New York, April/May 2002, pp. 29 – 44.

[30] S. D. Galbraith and J. McKee, *The Probablity that the Number of Points on an Elliptic Curve over a Finite Field is Prime*, Journal of the London Mathematical Society **62** (2000), no. 3, 671 – 684.

[31] R. Gallant, R. Lambert, and S. Vanstone, *Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms*, Advances in Cryptology - CRYPTO '91, Lecture Notes in Computer Science, vol. 2139, Springer–Verlag, Berlin, 2001, pp. 190–200.

[32] R. P. Gallant, R. Lambert, and S. A. Vanstone, *Improving the parallelized Pollard lambda search on binary anomalous curves*, Mathematics of Computation **69** (2000), 1699 – 1705.

[33] P. Gaudry, F. Heß, and N. P. Smart, *Constructive and Destructive Facets of Weil Descent on Elliptic Curves*, Journal of Cryptology **15** (2002), no. 1, 19.

[34] R. Granger, D. Page, and M. Stam, *On Small Characteristic Algebraic Tori in Pairing-Based Cryptography*, ePrint Archive, 2004.

[35] R. K. Guy, *Unsolved Problems in Number Theory*, 3rd ed., Problem Books in Mathematics, Springer–Verlag, 2004.

[36] D. Hankerson, J. López Hernandez, and A. J. Menezes, *Software Implementation of Elliptic Curve Cryptography Over Binary Fields*, Cryptographic Hardware and Embedded Systems, CHES'2000 (2000), 1 – 24.

[37] D. Hankerson, A. J. Menezes, and S. A. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer–Verlag, New York, 2005.

[38] A. G. H. V. F. A. A. Kerckhoffs von Nieuwenhof, *La Cryptographie Militaire*, Journal des Sciences Militaires **IX** (1883), 5–83.

[39] D. E. Knuth, *Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Third ed., vol. 2, Addison-Wesley Professional, 1997.

[40] N. Koblitz, *Elliptic Curve Cryptosystems*, Mathematics of Computation **48** (1987), 203 – 209.

[41] ———, *CM–curves with good cryptographic properties*, Advances in Cryptology - CRYPTO '91, Lecture Notes in Computer Science, vol. 576, 1992, pp. 279–287.

[42] S. Lang, *Algebra*, Third ed., Springer–Verlag, New York, 2005.

[43] A. K. Lenstra, *Using Cyclotomic Polynomials to Construct Efficient Discrete Logarithm Systems over Finite Fields*, ACISP97, LNCS, vol. 1270, Springer–Verlag, New York, 1997, pp. 123–138.

[44] A. K. Lenstra and H. W. Lenstra-Jr, *The Development of the Number Field Sieve*, Lecture Notes in Computer Science, Springer–Verlag, New York, 1993.

[45] A. K. Lenstra and E. R. Verheul, *An overview of the XTR public key system*, In Proceedings of the Warsaw Conference on Public-Key cryptography and computational number theory, 2000.

[46] ———, *The XTR Public Key System*, Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology, vol. 1880, Springer–Verlag, London, 2000, pp. 1–19.

[47] H. W. Lenstra-Jr, *Finding Isomorphisms Between Finite Fields*, Mathematics of Computation **56** (1991), no. 193, 329–347.

[48] H. Lidl and H. Niederreiter, *Introduction to finite fields*, Cambridge University Press, 1986.

[49] J. Luis-Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity I*, Monographs on Theoretical Computer Science, vol. 11, Springer–Verlag, 1988.

[50] ———, *Structural Complexity II*, Monographs on Theoretical Computer Science, vol. 22, Springer–Verlag, 1990.

[51] U. Maurer and D. Raub, *Black–Box Extension Fields and the Inexistence of Field–Homomorphic One-Way Permutations*, Advances in Cryptology - ASIACRYPT 2007: 13th International Conference on the Theory and Application of Cryptology and Information Security, Lecture Notes in Computer Science, vol. 4833, March 2007, pp. 427–443.

[52] A. Menezes and M. Qu, *Analysis of the Weil Descent Attack of Gaudry, Heß and Smart*, Lecture Notes In Computer Science, Conference on Topics in Cryptology: The Cryptographer's Track at RSA **2020** (2001), 308 – 318.

[53] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, Free Online ed., CRC Press, 1996.

[54] S. J. Miller and R. Takloo-Bighash, *An Invitation to Modern Number Theory*, Princeton University Press, April 2006.

[55] V. Miller, *Use of Elliptic Curves in Cryptography*, Advances in Cryptology - CRYPTO '85 (H. C. Williams, ed.), Lecture Notes in Computer Science, Springer–Verlag, New York, 1986, pp. 417–426.

[56] V. I. Nechaev, *Complexity of a determinate algorithm for the discrete logarithm*, Advances in Cryptology - Crypto '96 **1440** (1994), no. 2, 165–172.

[57] NIST, *Recommended Elliptic Curves for Federal Government Use*, National Institute of Standards and Technology (NIST), NIST Special Publication ed., July 1999.

[58] ———, *DIGITAL SIGNITURE STANDARD*, U.S. DEPARTMENT OF COMMERCE / National Institute of Standards and Technology, fips pub 186-2 ed., January 2000.

[59] L. Perret, *A Fast Cryptanalysis of the Isomorphism of Polynomials with One Secret Problem*, Advances in Cryptology – EUROCRYPT 2005, Lecture Notes in Computer Science, vol. 3494/2005, Springer Berlin / Heidelberg, May 2005, pp. 354 – 370.

[60] R. Pinch, *Recognising elements of finite fields*, Proceedings 2nd IMA Conference on Coding and Cryptography 1989 (C. J. Mitchell, ed.), vol. 33, Oxford University Press, 1991, pp. 193 – 197.

[61] S. Pohlig and M. Hellman, *An improved algorithm for computing logarithms over* GF(p) *and its cryptographic significance*, Information Theory, IEEE Transactions on **24** (1978), no. 1, 106 – 110.

[62] J. Pollard, *Monte Carlo methods for index computation mod p*, Mathematics of Computation **32** (1978), 918 – 924.

[63] M. O. Rabin, *Probabilistic algorithm for testing primality*, Number Theory **12** (1980), no. 1, 128 – 138.

[64] P. Ribenboim, *The Book of Prime Number Records*, Springer–Verlag, New York, 1988.

[65] H. Riesel, *Prime Numbers and Computer Methods for Factorization*, Progress in Mathematics, Birkhäuser, 1985.

[66] S. Roman, *Coding and Information Theory*, Graduate Texts in Mathematics, Springer–Verlag, New York, 1992.

[67] K. Rubin and A. Silverberg, *Torus Based Cryptography*, Advances in Cryptology (CRYPTO 2003), vol. Lecture Notes in Computer Science 2729, rberg. Torus-Based Cryptography. In Advances in Cryptology (CRYPTO 2003), Springer LNCS 2729, 349-365, 2003. 2003, pp. 349–365.

[68] ———, *Compression in finite fields and torus–based cryptography*, SIAM Journal on Computing **37** (2005), no. 5, 1401–1428.

[69] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*, Second ed., Wiley, 1995.

[70] R. Schoof, *Elliptic Curves over Finite Fields and the Computation of Square Roots modulo p*, Mathematics of Computation **44** (1985), 483–494.

[71] G. Seroussi, *Compact Representation of Elliptic Curve Points over* $\mathbb{F}_{2^n}$, Tech. report, HP Labs Tech. Report HPL-98-94R1, September 1998.

[72] ———, *Table of Low-weight Binary Irreducible Polynomials*, Tech. report, HP Labs Tech. Report HPL-98-135, August 1998.

[73] V. Shoup, *Lower Bounds for Discrete Logarithms and Related Problems*, Advances in Cryptology - EUROCRYPT '97, Lecture Notes in Computer Science, vol. 1233, Springer–Verlag, Berlin, 1997, pp. 256–266.

[74] J. H. Silverman, *Advanced Topics in the Arithmetic of Elliptic Curves*, Springer–Verlag, 1995.

[75] ———, *The Arithmetic of Elliptic Curves*, Second ed., Graduate Texts in Mathematics, Springer–Verlag, New York, 1997.

[76] N. P. Smart, *Compressed ECC Parameters*, SEC Group report (2000), 1–3.

[77] ———, *Cryptography: An Introduction*, First ed., McGraw-Hill Education, 2003.

[78] N. P. Smart, D. Brown, A. W. Dent, E. Oswald, M. Joye, F. Vercauteren, P. Gaudry, F. Heß, S. Galbraith, and K. G. Paterson, *Advances in Elliptic Curve Cryptography: Further Topics Vol. 2*, First ed., London Mathematical Society, Lecture Note Series, no. 317, Cambridge University Press, 2005.

[79] P. Smith and M. J. J. Lennon, *LUC: A new public key system*, Tech. report, LUC Partners, Auckland UniServices Ltd, The University of Auckland, 1993.

[80] J. A. Solinas, *Generalized Mersenne Numbers*, Technical report CORR-39, University of Waterloo (1999), 1–23.

[81] ———, *Improved algorithms for arithmetic on anomalous binary curves*, Tech. Report 96–46, Combinatorics and Optimization Research Report, University of Waterloo, Canada, 1999.

[82] ———, *Efficient Arithmetic on Koblitz Curves*, Designs, Codes and Cryptography **19** (2000), no. 2–3, 195–249.

[83] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, Second ed., Cambridge University Press, 2003.

[84] J. von zur Gathen and M. Nöcker, *Polynomial and Normal Bases for Finite Fields*, Journal of Cryptology **18** (2005), no. 4, 337–355.

[85] Z. Wan, *Lectures on Finite Fields and Galois Rings*, World Scientific, 2003.

[86] L. C. Washington, *Elliptic Curves: Number Theory and Cryprography*, First ed., Discrete Mathematics and its Applications, Chapman & Hall/CRC Press, 2003.

[87] M. J. Wiener and R. J. Zuccherato, *Faster Attacks on Elliptic Curve Cryptosystems*, Lecture Notes In Computer Science, SAC '98 (1999), no. 1556, 190–200.

[88] A. Woodbury, D.V. Bailey, and C. Parr, *Elliptic Curve Cryptography on Smart–cards without coprocessor*, Smart Card Research and Advanced Applications - CARDIS, vol. 180, 2000, pp. 71–92.

[89] J. W. Wrench, *Evaluation of Artin's Constant and the Twin Prime Constant*, Mathematics of Computation **15** (1961), 396–398.