Applications of Frobenius Expansions in Elliptic Curve Cryptography

Waldyr Dias Benits Junior

Technical Report
RHUL–MA–2009–12
4 March 2009

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
http://www.rhul.ac.uk/mathematics/techreports

Applications of Frobenius Expansions in Elliptic Curve Cryptography

by

Waldyr Dias Benits Júnior

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

Department of Mathematics
Royal Holloway, University of London
2008

Declaration

These doctoral studies were conducted under the supervision of Professor Steven D. Galbraith.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Department of Mathematics as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

London, UK, 03 September 2008.

Waldyr Dias Benits Júnior

Abstract

Recent developments in elliptic curve cryptography have heightened the need for fast scalar point multiplication, specially when working on environments with limited computational power. It is well known that point multiplication on elliptic curves over $\mathbb{F}_{q^m}$ (with $m > 1$) can be accelerated using Frobenius expansions. In practice, the computation is much faster than the standard double-and-add scalar multiplication.

An efficient implementation of elliptic curve cryptosystems can use a Koblitz curve and convert integers into Frobenius expansions to perform fast scalar multiplications. However, this conversion of integers to Frobenius expansions would lead to extra code on the device (i.e., silicon area) and extra computational cost.

According to N. Koblitz, H. Lenstra suggested that rather than choosing a random integer n and then converting to a Frobenius expansion $n(\tau)$, in certain cryptosystems it might be more efficient to generate a random Frobenius expansion directly. The temptation then is to choose a relatively short and/or sparse value for $n(\tau)$. If this is done then we must re-evaluate the difficulty of the discrete logarithm problem (and other computational problems). A further issue is that the existing security proofs may not directly apply. For some systems it may be necessary to develop bespoke security proofs for the Frobenius expansion case.

In this thesis, we analyse the Frobenius expansion DLP and present algorithms to solve it. Furthermore, we propose a variant of a well known identification scheme designed for public key cryptography on very restricted devices. More precisely, we construct the Girault-Poupard-Stern (GPS) identification scheme for Koblitz elliptic curves using Frobenius expansions. The idea is to use Frobenius expansions throughout the protocol, so there is no need to convert between integers and Frobenius expansions. We also give a security analysis of the proposed scheme.

Acknowledgements

I would like to give special thanks to my Supervisor Professor Steven D. Galbraith for his valuable help, countless patience (specially with my English, in the beginning of my research), for being available whenever needed and also for all his suggestions and ideas that made this thesis come true.

I would like to thank the Lord, for giving me health and courage to be able to finish this work.

I would like to express my gratitude to my beloved wife Luciana and lovely daughter Gabriella, for always being supportive, believing in me (even when I did not believe in myself) and for their understanding about the countless hours that I dedicated to my project when I could not be with them.

I would like to thank my mother Vilma for the endless love she always gave me, for everything she taught and also for being Mum and Dad since Dad passed away.

I would like to thank my father Waldyr for always standing by me, leading the way and rooting for me. Dad, wherever you are, I know that you now must be close to God and very proud of me.

My acknowledgements also go to Raminder for his remarks on some chapters of this thesis and for the examiners, Dr. Roberto Avanzi and Dr. James McKee, who helped to improve this thesis with valuable comments.

Last, but not least, I would like to thank the Brazilian Navy for footing the bill during the three years I stayed abroad.

"For thousands of years, kings, queens and generals have relied on efficient communication in order to govern their countries and command their armies. At the same time, they have all been aware of the consequences of their messages falling into the wrong hands, revealing precious secrets to rival nations and betraying vital information to opposing forces. It was the threat of enemy interception that motivated the development of codes and ciphers: techniques for disguising a message so that only the intended recipient can read it."

Simon Singh

Contents

List of Abbreviations

We list now the abbreviations used throughout this work. The number indicates the page where the entry first appears.

τ-DLP	Frobenius Expansion DLP	166
τ-GPS	GPS protocol using Frobenius Expansions	166
τ-NAF	Frobenius Expansion in Non-Adjacent Form	75
CCA1	Chosen Ciphertext Attack (Lunchtime Attack)	27
CCA2	Adaptive Chosen Ciphertext Attack	27
BSGS	Baby-Step Giant-Step	44
DLP	Discrete Log Problem	40
ECDLP	Elliptic Curve Discrete Log Problem	43
ECC	Elliptic Curve Cryptography	43
GPS	Girault – Poupard – Stern Identification Scheme	72
IND	Indistinguishability	26
NAF	Non Adjacent Form	75
OWE	One-way Encryption	26
ROM	Random Oracle Model	29
RSA	Rivest – Shamir – Adleman	166
ZK	Zero Knowledge	31

List of Definitions

We list now the definitions used throughout this work. The number indicates the page where the definition appears.

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, there has been an increasing interest in environments with limited computational power. It is well known that point multiplication on elliptic curves over $\mathbb{F}_{q^m}$ (with $m > 1$) can be accelerated using Frobenius expansions (a.k.a. τ-adic expansion). H. Lenstra suggested that rather than choosing a random integer n and then converting to a Frobenius expansion $n(\tau)$, in certain cryptosystems it might be more efficient to generate a random Frobenius expansion directly. The temptation then is to choose a relatively short and/or sparse value for $n(\tau)$.

The GPS identification protocol was the starting point of our entire project. At the beginning, we were tempted to use Frobenius expansions with short exponents in the GPS protocol, which would lead to smaller parameters, compared with the usual integer case, because we believed, at that point, that low-memory counterparts for the Frobenius expansions discrete log problem (which we called τ-DLP) did not exist, or at least, did not have a square-root behaviour.

By studying the τ-DLP and with some suggestions by Tanja Lange, we discovered that low memory algorithms with square root complexity for the τ-DLP do exist, although with non-optimal constants hidden in the $O(\ )$.

1.1 Road map

We begin this thesis with a background chapter, in that we review some topics that we judged important for a better understanding of our work. The notations (standard notations, whenever possible) introduced in Chapter 2 are used throughout the thesis.

The next stop of our thesis is also a background chapter, but we decided to present it separately, with more details, due to its relevance in relation to our work. In Chapter 3 we study the elliptic curve discrete log problem (ECDLP) and present the standard algorithms to solve them. We briefly review the exhaustive search algorithm and the Shanks' baby-step giant-step (BSGS) algorithm. Then we focus on the low memory algorithms to solve the ECDLP, when we analyse the Pollard-rho and the Pollard-kangaroo methods, as well as their parallel counterparts. The understanding of Pollard-kangaroo method will be paramount for the study of a low memory algorithm for the τ-DLP.

In Chapter 4 there is also some review, as well as new material. We revise the concept of Koblitz curves and Frobenius expansions, the latter being the actor in a leading role of this thesis. We show how to efficiently compute a scalar point multiplication using Frobenius expansions, some subtleties of the arithmetic with τ-adic expansions and also that every Frobenius expansion can be mapped down to $x + y\tau$, for integers x, y of certain size. As far as we know, we are the first to prove the bounds for x and y and also conjecture that such bounds can be shortened, based on the distribution of τ-adics. The use of this property will be the main tool to solve the τ-DLP. In this chapter, we also present an algorithm for adding two τ-adic expansions, and show that we needed to include a randomisation step in order to avoid a statistical attack in the τ-GPS protocol.

Chapter 5 is dedicated to the Frobenius expansion DLP. We observed that a deep study of the τ-DLP was necessary, in order to guide the use of Frobenius expansions in cryptosystems. We introduce three computational problems, namely, the general τ-DLP, the τ-NAF DLP and the weight-w τ-DLP. Similarly with what we have done in Chapter 3, we briefly present the exhaustive search and BSGS algorithms for the three computational problems and then we concentrate on a low-memory algorithm. The solution was to use the Gaudry-Schost 2-dimensional algorithm and use the property that a τ-adic can be written as $x + y\tau$, for integers x, y. We give a complete description of Gaudry-Schost algorithm and analyse it, trying to fill some gaps in the work of Gaudry and Schost.

After a better understanding of the τ-DLP, the next stop of our thesis is to present an application of a real protocol using Frobenius expansions in the place of integers. In Chapter 6 we study the GPS identification scheme and propose a counterpart using Frobenius expansions, which we called τ-GPS. Since the study of τ-DLP showed that we cannot use shorter parameters, the main advantage of using Frobenius expansions is that it does not require conversion between integers and Frobenius expansions, being useful to applications with limited offline computation time and limited code area. We give a security analysis of τ-GPS and also some hints of how to choose the right protocol for constrained devices, according to the application.

In Chapter 7 we define parameters sizes needed for efficient and secure applications of Frobenius expansions, based on the results of Chapters 5 and 6.

Finally, Chapter 8 concludes our work and present some open problems and suggestions of future work.

1.2 Contributions

Now we highlight the main contributions of this thesis, in sequential order.

- [Section 4.2.2] Let L be a parameter which represents the length of a τ-adic expansion $\sum_{i=0}^{L-1} \alpha_i \tau^i$ with coefficients $\alpha_i \in \{-1, 0, 1\}$. The number of possible τ-adic expansions is clearly 3^L. However, we have many P-equivalent τ-adics in this set, i.e., Frobenius expansions a and b such that $[a]P = [b]P$ for some point $P \in E(\mathbb{F}_{2^m})$, which results in a much smaller number of equivalence classes. We compute a bound for the number of P-equivalence classes of Frobenius expansions. We experimentally show that this number is bounded by $\tilde{O}(2^L)$;

- [Sections 4.2.4, 4.2.6 and 4.2.6.2] Any τ-adic expansion of the form $\sum_{i=0}^{L-1} \alpha_i \tau^i$ can be mapped down to $x + y\tau$, for integers x and y of certain size. We prove bounds on x and y and also we give some pictures of the distribution of random τ-adics and random τ-NAFs. From these pictures, we conjecture optimal bounds on x and y;

- [Section 4.3] We propose the use of a randomisation step in the algorithm to add and multiply τ-adics, in order to avoid a statistical attack in the τ-GPS protocol;

- [Section 5.2] We introduce three new computational problems, namely, the general τ-DLP, the τ-NAF DLP and the weight-w τ-DLP;

- [Section 5.5.2] We fill some gaps in the analysis of the Gaudry Schost algorithm, giving an estimate of how many steps are likely to be outside of the search box and analysing the success probability of the algorithm;

- [Section 6.5] We propose a GPS protocol which uses Frobenius expansions throughout (and we call it τ-GPS). This will lead to fast and simple offline operations while still keeping the online operation fast, being useful to applications with limited offline computation time and limited code area;

- [Section 6.6] We give a security analysis for the proposed τ-GPS;

- [Section 6.6.3.1] We describe a statistical attack on the τ-GPS protocol if the randomisation step proposed in Section 4.3 is not used;

- [Section 7.5] Let $\mathcal{C}$ be the length of the τ-adic which represents the challenge in the GPS protocol and let $\mathcal{S}$ be the length of the τ-adic which represents the private key. We compute the number of possible τ-adic expansions of degree less than $\mathcal{C}$, Hamming weight w and at least $\mathcal{S}-1$ zero coefficients between each pair of nonzero coefficients, which will be useful if one wants to use the Girault-Lefranc trick with τ-GPS.

Chapter 2

Background

Contents

In this chapter we present a review of topics which are important for the comprehension of the thesis. Any reader with some basic knowledge of the topics presented here can skip this chapter without any risk of loss of understanding, since nothing original will be presented. We also give some references for those who want to obtain further knowledge about the topics presented in this chapter.

2.1 Complexity

The first step of our review is to present some notation and definitions well known in the literature, which will be used throughout this thesis. The term or expression to be defined will be written in **bold**. The definitions are based on those given in [17, 37, 54].

Definition 1. *An* ***algorithm*** *is a finite sequence of steps to solve a problem.*

Let n be the input size of an algorithm $\mathcal{A}$ and let $f(n)$ be a function which represents the running time of $\mathcal{A}$ in its worst case (in other words, the running time of $\mathcal{A}$ is upper bounded by $f(n)$).

Definition 2. *We say that $f(n) \in O\big(g(n)\big)$ (a.k.a.* ***big-O notation****) if there exists some constant $c > 0$ and some integer n_0 such that $0 \leqslant f(n) \leqslant cg(n)$ for all $n > n_0$.*

Definition 3. *We say that $f(n) \in \tilde{O}\big(g(n)\big)$ (read* **soft**-O notation*) if $f(n) = O(g(n)\log^k(g(n)))$ for some k. In other words, it means that logarithmic factors are ignored in the big-O notation.*

Definition 4. *We say that $f(n) \in \Omega\big(g(n)\big)$ if there exists some constant $c > 0$ and some integer n_0 such that $0 \leqslant cg(n) < f(n)$ for all $n > n_0$.*

Definition 5. *Informally speaking, we say that algorithm $\mathcal{A}$ runs in:*

- ***polynomial time*** *if there exists some constant k such that $f(n) \in O(n^k)$;*
- ***sub-exponential time*** *if for all k, we have $f(n) \in \Omega(n^k)$ and for all constants $a > 1$ we have $f(n) \in O(a^n)$;*

- ***exponential time*** *if there exists some constants* $a, b > 1$ *such that* $f(n) \in \Omega(a^n)$ *and* $f(n) \in O(b^n)$.

For constants c *and* v*, we have:*

$$L_p(v, c) = \exp\left(c(\lg p)^v (\lg \lg p)^{1-v}\right),$$

where if:

$$\begin{cases} v = 1, & L_p \text{ is exponential in } \lg p; \\ v = 0, & L_p \text{ is polynomial in } \lg p; \\ 0 < v < 1, & L_p \text{ is sub-exponential in } \lg p. \end{cases}$$

Definition 6. *A parameter (for example, a probability)* $\varepsilon : \mathbb{N} \to \mathbb{R}$ *is **negligible** if for any nonzero polynomial* $\mathcal{P}$*, there exists* m *such that*

$$\forall n > m, |\varepsilon(n)| < \frac{1}{\mathcal{P}(n)} \ .$$

On the other hand, a probability Pr *is **overwhelming** if* $1 - Pr$ *is negligible.*

Definition 7. *For inputs of size* n*, if no polynomial time adversary can, with non-negligible probability, successfully solve some problem, we call this problem **hard**.*

2.2 Cryptography

The term *cryptography* – derived from Greek *cryptos* (hidden) and the verb *graphein* (to write) – means "secret writing". According to [54], "the fundamental objective of cryptography is to enable two people to communicate over an insecure channel in such a way that an adversary cannot understand what is being said".

In order to establish a secure communication, we need to provide the following services [37, 52]:

Confidentiality (a.k.a. privacy) means to ensure that all information (stored or transmitted) has been revealed only to authorised users;

Authenticity means to ensure that all parts involved in a communication have been correctly identified;

Integrity means to ensure that all information (stored or transmitted) has not been altered by any unauthorised user.

Non-Repudiation means to ensure that neither the sender nor the receiver of some information can deny, later, that he or she has transmitted or received such information.

Access Control means to ensure that any resource cannot be accessed by unauthorised users.

Availability means to ensure that any resource be available to authorised users whenever they need it.

2.3 Public Key Cryptography

For centuries, cryptography has been used to provide privacy of communication. Besides privacy, it is also important to authenticate one part to another. Before 1976, privacy and mutual authentication were achieved with *Symmetric Cryptography*, where each pair of users share a secret key. The main problems of symmetric cryptography are the great number of keys needed in a communication among n users and the need of a secure channel to transmit the secret keys. Furthermore, symmetric cryptography does not provide the non-repudiation service.

The concept of Public Key Cryptography (PKC) first appeared in 1976, in Diffie and Hellman's paper "*New Directions in Cryptography*"[20]. In that paper, the authors proposed an *asymmetric cryptography* model in that, unlike symmetric cryptography, each user has a key pair, say (S_k, P_k), where S_k, called the *private key*, is a secret known only by his owner and the other key P_k, known as the *public key*, is kept in a public domain. Usually, each user generates his own key pair, one of them random (S_k) and the other (P_k) computed as a function of S_k. It is well known, nowadays, that a public key must be associated to a *digital certificate* issued by a trusted authority, which links the key to its owner.

As usual, we ask for the help of two well-known characters in cryptography, named as *Alice* and *Bob* to exemplify the concept of public key cryptography. Let (S_A, P_A) and (S_B, P_B) be, respectively, the key pairs of Alice and Bob. We assume that both P_A and P_B have a valid digital certificate, signed by a trusted authority. If Alice wants to send a message m (a.k.a. *plaintext*) to Bob using public key cryptography, she first finds Bob's public key in a public domain, then she encrypts m using P_B and finally, she sends the resultant *ciphertext* c to Bob. After receiving c, Bob uses his private key and recovers m. It is easy to notice that any person could play the role of Alice (this is known as *impersonation*). If Bob needs to ensure that the person who sent him a message is, in fact, Alice, a *digital signature* is needed. We will see digital signature in detail in Section 2.8.

Let M_k be the space of all possible messages; PK_k be the space of all possible public keys; SK_k be the space of all possible private keys; and C_k be the space of all possible ciphertexts. Public key encryption can be formally defined under a security parameter k with the following algorithms:

KeyGen a randomised algorithm which takes k as input and outputs $pk \in PK_k$ and $sk \in SK_k$;

Encrypt a randomised algorithm which takes pk and $m \in M_k$ as input and outputs $c \in C_k$ in polynomial time in 2^k;

Decrypt an algorithm (usually deterministic) which takes sk and $c \in C_k$ as input and outputs either $m \in M_k$ or $\perp$ (invalid ciphertext symbol) in polynomial time in 2^k.

The key pair (pk, sk) is a valid key pair if:

$$Decrypt(Encrypt(m, pk), sk) = m.$$

We now present the security properties for an encryption scheme:

One way encryption (OWE): An adversary is not able to compute m if he has access only to the corresponding ciphertext c;

Semantic security: An adversary learns no information at all about m (except possibly its length) from the corresponding ciphertext c;

Indistinguishability (IND): An adversary is not able to distinguish the encryption of two messages m_0 and m_1 of the same length.

Precisely, an indistinguishability adversary is an algorithm $\mathcal{A}$ which in the first stage takes pk as input and outputs messages m_0 and m_1 of the same length. In the second stage, $\mathcal{A}$ receives the *challenge ciphertext* c (where $c = Encrypt(m_b)$, with $b \in \{0, 1\}$ randomly chosen) as input and outputs b'.

Let $Pr_{[b=b']}$ be the probability that an adversary outputs $b = b'$ and let $|Pr_{[b=b']} - \frac{1}{2}|$ be the advantage of an adversary. If no adversary has non-negligible advantage an encryption scheme is said to have the indistinguishability property.

The following attack models are known in public key encryption:

Passive attack An adversary has access to the public key only.

Lunchtime attack (CCA1) An adversary has access to the public key and can also ask for decryptions of his chosen ciphertexts (before receiving the challenge ciphertext).

Adaptive chosen-ciphertext attack (CCA2) An adversary has access to the public key and to a decryption oracle which outputs decryptions of any chosen ciphertext, except the *challenge* ciphertext.

We define **standard model** as the model of computation in which the adversary is only limited by the amount of time and computational power available. The strongest notion of security for encryption schemes is known to be indistinguishability under a CCA2 attack in the standard model.

We refer to [20, 37, 54] for further knowledge about public key cryptography.

2.4 Hash Functions and Random Oracles

A *hash function* is a mathematical function which maps an entry x of any size to an output y of fixed size. In order to be used in cryptography, a hash function ideally must satisfy the following properties:

1. **Compression** - h maps an input x of any size to an output y of fixed size;
2. **Ease of computation** - given x, one can compute $y = h(x)$ in polynomial time;
3. **Preimage resistance** - given y, it should be *hard* to compute x such that $h(x) = y$;
4. **Second preimage resistance** - given x, it should be *hard* to compute $z \neq x$ such that $h(x) = h(z)$;
5. **Collision resistance** - it should be *hard* to find any values x and z such that $h(x) = h(z)$.

See Definition 6 in Section 2.1 for a precise explanation of "hard".

Many cryptographic schemes make use of hash functions. However, sometimes it is difficult to write a formal proof of security for protocols using a real hash function. Thus, many authors suggest the use of an ideal (theoretical) computational model, in which they can ignore the internal structure of hash functions and can concentrate in the protocol itself. In such an ideal model, the hash functions are replaced by random oracles, defined by [7] as follows:

"A Random Oracle R is a map from $\{0,1\}^*$ to $\{0,1\}^\infty$ chosen by selecting each bit of $R(x)$ uniformly and independently, for every x. Of course, no actual protocol uses an infinitely long output, this just save us from having to say how long 'sufficiently long' is."

In other words, the random oracle models a hash function as a random function. In a proof of security, we simulate the output of a random oracle using binary sequences which are indistinguishable from a random sequence.

Cryptographic schemes which can be proven secure using only complexity assumptions are said to be secure in the standard model, whereas schemes which use random oracles are said to be secure in the random oracle model (ROM).

We refer to [37, 54] for further details about hash functions.

2.5 Elliptic Curves

We review here only some of the most important facts about elliptic curve cryptography. More details can be found in [2, 9, 23, 37, 38, 48, 49, 54].

Let $p \geqslant 2$ be a prime number and let $q = p^m$, for some $m \in \mathbb{N}$. Let $\mathbb{F}_q$ (also written as $GF(q)$) be a finite field of q elements. If q is prime, we can think of $\mathbb{F}_q$ as the set of integers modulo q ($\mathbb{Z}_q$). Let $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_q$. We say that the *elliptic curve over* $\mathbb{F}_q$ is the set of solutions $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$ for the **Weierstrass equation**

$$E : y^2 + a_1xy + a^3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.1}$$

together with a special point $\mathcal{O}$, called the *point at infinity*.

After a change of variables, Equation (2.1) can be simplified to the following forms (known as **simplified Weierstrass form** for curves of characteristic p):

$$E : \begin{cases} y^2 + xy = x^3 + ax^2 + b & \text{if } p = 2 \\ y^2 = x^3 + ax^2 + bx + c & \text{if } p = 3 \\ y^2 = x^3 + ax + b & \text{if } p > 3 \end{cases}$$

When $p = 2$ we need $b \neq 0$, when $p = 3$ we need $a^2(b^2 - 4ac) - b^3 \neq 0$ and when $p > 3$ we need $4a^3 + 27b^2 \neq 0$ to ensure that E has no multiple roots, so it is possible to draw a tangent line in any point of the curve.

It is well known that the points of an elliptic curve define a group law, with $\mathcal{O}$ as the identity element.

We now define some useful concepts in elliptic curves:

- point multiplication (a.k.a. scalar point multiplication): let s be an integer and P an elliptic curve point. We define $[s]P$ as the sum of P with itself s times. There are well defined formulae for adding two points P and Q of an elliptic curve ($R = P + Q$) or for computing the scalar point doubling. We refrain to give such formulae here, but they can be easily found in the references we gave in the beginning of this section. We remark that $[s]P$ can be efficiently computed (i.e., computed in polynomial time) using double-and-(add or subtract) algorithm (see Section 4.2.3 and also [54, pg 266]).
- curve order (#E): is the number of points of a given curve E;

- order of a point P: is the smallest integer d such that $[d]P \equiv \mathcal{O}$.

The Elliptic Curve Discrete Log Problem (ECDLP) will be treated separately in Chapter 3.

2.6 Zero Knowledge

The concept of Zero knowledge (ZK) was introduced by Goldwasser *et al* [30]. According to Goldreich [29]: "zero knowledge proofs are proofs that are both convincing and yet yield nothing beyond the validity of the assertion being proved". In other words, anything that can be extracted (computed) from a zero knowledge proof, can also be computed from the assertion itself.

We begin this section giving a mathematical example which illustrates a zero-knowledge proof. Let $G(V, E)$ be a graph simple and connected. Let n be the number of vertices of G. We say that a G is 3-colourable if there is a map $\phi : V \to \{A, B, C\}$ — where $\{A, B, C\}$ represent three different colours (e.g., A = Red, B = Blue, C = Green) — such that every two adjacent vertices are assigned different colours. In other words, for each $(u, v) \in E$, we have $\phi(u) \neq \phi(v)$. Finding ϕ is equivalent to partitioning the set of vertices of G into three independent subsets, V_1, V_2 and V_3, such that each subset contains no adjacent vertices. See Figure 2.1 for an example of a 3-coloured graph.

We suppose that a prover, say Peggy, claims to a verifier, say Victor, that she knows how to 3-colour G. Peggy wants to convince Victor that she really knows it, without revealing to Victor how to 3-colour G. We assume

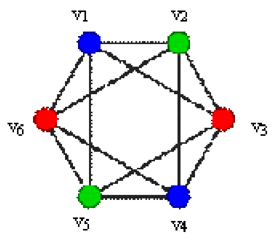

Figure 2.1: The graph above is 3-coloured. Notice that the map ϕ divides the set of vertices into 3 independent subsets: $V_1 = \{v_1, v_4\}$, $V_2 = \{v_2, v_5\}$ and $V_3 = \{v_3, v_6\}$.

that Peggy has n boxes, labelled with the integers $1, 2, 3, \ldots, n$ and each box has a lock with a corresponding and unique key.

Peggy and Victor run the following protocol:

- Peggy chooses at random a permutation of $\{A, B, C\}$ (i.e., if Peggy, without loss of generality, chooses $\{B, C, A\}$, vertices in subset V_1 are coloured with *blue*, vertices in V_2 are coloured with *green* and vertices in V_3 are coloured with *red*), fills each of the n boxes with the assigned colour (i.e., if vertex v_i is coloured blue, Peggy will put a "blue card" inside box i), locks all the boxes and sends them (without keys) to Victor;

- Victor chooses at random a pair of adjacent vertices (v_j, v_k) in G (the *challenge*) and sends it to Peggy;

- Peggy verifies if (v_j, v_k) are adjacent vertices in G and sends the keys corresponding to boxes j and k to Victor. If (v_j, v_k) are non adjacent vertices, Peggy does nothing;

- Victor opens the boxes j and k and verifies if they contain two different colours, among $\{A, B, C\}$. In case Victor finds a colour not in $\{A, B, C\}$ (e.g., Victor finds a yellow card in box v_j), or the same colour in the two boxes or even if the keys sent by Peggy do not open the corresponding box, Victor rejects the proof and stops the protocol.

Of course, if Peggy correctly guesses Victor's challenge (i.e, (v_j, v_k)), she succeeds even if she does not know how to 3-colour G. However, if they repeat this protocol many times, her chance of successfully anticipating all of Victor's requests becomes vanishingly small, and Victor should be convinced that she knows the secret.

Now, after giving an idea of a zero-knowledge protocol, we continue with some definitions:

A *proof of knowledge* is defined as an interactive proof in which the prover successfully convinces a verifier that he knows some secret. A proof of knowledge, among many other applications, can be used to construct identification protocols or signature schemes.

A *zero-knowledge* proof (a.k.a. zero-knowledge protocol) is a proof of knowledge in that no information at all about the secret is revealed to the verifier (except the fact that the prover really knows such secret). For instance, the verifier (after being convinced that the prover does know the secret) cannot convince a third party the he or she knows the secret as well. Back to the 3-colouring example, Victor is convinced that Peggy knows the secret, but he learns nothing else except that Peggy is telling the truth. Furthermore, despite being convinced that Peggy knows ϕ, Victor has no means to convince a third party that he knows how to 3-colour G.

We define a prover as *honest* if he or she knows the secret and *dishonest* otherwise; we say that a verifier is *honest* if he or she runs the protocol properly and *dishonest* if otherwise (in this case, the dishonest verifier cheats during the protocol, trying to obtain some information about the secret). A zero-knowledge proof must satisfy three properties:

- **Completeness**: an honest prover always[1] succeeds to convince an honest verifier that he or she (the prover) knows the secret.
- **Soundness**: A dishonest prover can only convince an honest verifier that he or she (the prover) knows the secret with negligible probability.
- **Zero-knowledge**: a prover, when interacting with any verifier (honest or not) does not leak any information about the secret, except the fact that he or she does know the secret. This is formalized by showing that there exists some simulator (a polynomial time algorithm) that, given only the statement to be proven (and no interaction with the prover), outputs an answer which is indistinguishable from the answer given by an interaction between the verifier and the real prover.

The first two of these are properties of more general interactive proof systems. The third is what makes the proof zero-knowledge.

Depending on what "indistinguishable from the answer given by an interaction between the verifier and the real prover" really means, some variants of zero-knowledge can be defined:

Perfect zero-knowledge if the distributions produced by the simulator and the real protocol are exactly the same.

[1]To be precise, we can say that the prover convinces the verifier with *overwhelming* probability

Statistical zero-knowledge if the distributions are not necessarily exactly the same, but they are statistically close, meaning that their statistical difference is a negligible function. Formally, it means that no algorithm which given a polynomial number of outputs of both distributions (simulator and real interaction with the prover) can distinguish, with non-negligible advantage, which of the two distributions an output belongs to, even when unlimited computational power is available.

Computational zero-knowledge if no efficient algorithm can distinguish the two distributions, i.e., an adversary (with computationally bounded power) cannot distinguish between the simulation and runs of the real protocol.

Further knowledge can be found in [22, 37].

2.7 Identification Protocols

We recall from Section 2.2 that the authentication service requires that all parties involved in a communication have been correctly identified. Some authors (for example, [37, 54]) interchangeably use the terms *identification* or *entity authentication* when talking about the authentication service. However, it can be found in the literature that these terms may have a different meaning, in that authentication is a process which involves identification plus verification, where identification is the answer to the question "Who are you ?" and verification is the answer to the question "Can you prove it ?". In this thesis, we use the former definition, i.e., we use identification and entity authentication as synonyms.

We will see in Section 2.8, for example, that Alice can prove her identity using her private key to sign a message. Digital signatures provide message authentication. However, Alice's identification can be done with an operation known as *identification* (or *entity authentication*) protocol. According to Menezes *et al* [37]:

"A major difference between entity authentication and message authentication (as provided by digital signatures) is that message authentication itself provides no timeliness guarantees with respect to when a message was created, whereas entity authentication involves corroboration of a claimant's identity through actual communications with an associated verifier during execution of the protocol itself (i.e., in *real-time*, while the verifying entity awaits). Conversely, entity authentication typically involves no meaningful message other than the claim of being a particular entity, whereas message authentication does."

Roughly speaking, we can say that identification protocols are performed in "real time" (i.e., the presence of the prover is required during the protocol), whereas message authentication, such as digital signatures, can be done any time after the message has been signed (i.e., the signature verification is usually performed without the presence of the signer).

We refer Menezes *et al* [37] again to define the main objectives of identification protocols:

1. If $\mathcal{A}$ and $\mathcal{B}$ are honest parties, the protocol between $\mathcal{A}$ and $\mathcal{B}$ is always successful, i.e., $\mathcal{B}$ will accept the authentication of $\mathcal{A}$;

2. (*transferability*) $\mathcal{B}$, after successful authenticating $\mathcal{A}$, cannot reuse any knowledge to impersonate $\mathcal{A}$ to a third party $\mathcal{C}$;

3. (*impersonation*) A third party $\mathcal{C} \neq \mathcal{A}$ carrying out the protocol with $\mathcal{B}$ and playing the role of $\mathcal{A}$ is only accepted by $\mathcal{B}$ with negligible probability (we recall that a formal definition of "negligible" was given in Section 2.1);

According to [37], "The previous points remain true even if: a (polynomially) large number of previous authentications between $\mathcal{A}$ and $\mathcal{B}$ have been observed; the adversary $\mathcal{C}$ has participated in previous protocol executions with either or both $\mathcal{A}$ and $\mathcal{B}$; multiple instances of the protocol, possibly initiated by $\mathcal{C}$, may be run simultaneously".

See also D. Sitnson [54] for more details.

2.8 Digital Signatures

We remember from Section 2.3 that, when Bob receives a message m encrypted with his public key, he cannot be sure that m was sent by Alice, unless she uses her private key to sign m before encrypting and sending it to Bob. In this case, Bob will reverse the process, using his private key to decrypt the ciphertext and then using Alice's public key to verify her signature. We recall that Bob can verify Alice's signature without her presence. Alice can also choose to encrypt m first and then she signs the correspondent ciphertext and send it to Bob. In this case, Bob first uses Alice's public key to verify her signature and then he uses his own private key to decrypt the ciphertext and obtains m.

A signature scheme consists of two main steps:

Sign A randomised algorithm which takes as input a private key (S_k) and a message (m) and outputs a signature (s).

Verify A (usually deterministic) algorithm which takes as input the public key (P_k) associated with signer's private key, a message (m), a signature (s) and outputs either "accept" or "reject".

Now we describe the main adversarial goals for digital signatures:

Total Break An adversary can obtain the private key, so he is able to forge a signature on any message.

Selective Forgery An adversary, with non-negligible probability, can create a valid signature on a chosen message.

Existential Forgery An adversary can create a valid pair (m, s), where s is the signature of m.

The following attack models are known in digital signatures:

Passive attack An adversary has access to the public key only.

Known message attack An adversary has access to a (finite) number of valid message-signature pairs.

Adaptive chosen-message attack An adversary has access to a signing oracle, which generates valid signatures for messages of his choosing (except the one which he is trying to forge a signature).

So, security against existential forgery under adaptive chosen message attack can be defined as the strongest notion of security for signatures.

We point out that the security of a signature scheme depends on the hardness of some mathematical problem. For example, the RSA-signature depends on the hardness of integer factorization.

2.9 Schnorr Identification and Signature

Now that we have revised the concepts of zero knowledge, identification protocols and signature schemes, we use the Schnorr scheme [46] as an example of the evolution of such concepts:

Previously, a trusted third party T chooses an element $g \in \mathbb{Z}_q^*$ of prime order $\mathbf{r}$ dividing $q-1$. Alice holds a valid key pair (s, I), where $s : 1 \leqslant s < \mathbf{r}$ is her private key and $I = g^{-s} \pmod q$ her public key.

Suppose that Alice wants to convince Bob that she knows s. Obviously, she could show s to Bob, but Alice does not want to reveal her secret. Then, she starts the following protocol with Bob:

In the three step protocol, Alice generates a secret random $r : 1 \leqslant r < \mathbf{r}$, computes the **commitment** $X = g^r \pmod q$ and sends X to Bob. Note that many "X" can be pre-computed by Alice. Then, Bob generates a random **challenge** $c : 1 \leqslant c \leqslant 2^t$ and sends it to Alice. The parameter t is called the security parameter. After receiving c, and checking its size, she computes the **response** $y = r + sc \pmod{\mathbf{r}}$ and sends y to Bob. The computation of y is the only online step. Finally, Bob checks whether or not $X \equiv g^y I^c \pmod q$. If $X \not\equiv g^y I^c \pmod q$, Bob rejects the proof. This round can be repeated l times to convince Bob that Alice is not only a lucky girl, she really knows s.

More precisely, the verification step is:

$$\begin{aligned} X &= g^y I^c \pmod{q} \\ &= g^{r+sc} g^{-sc} \\ &= g^{r+sc-sc} \\ &= g^r \end{aligned}$$

$\square$

Since the only value generated by Alice in the verification step is $y = r + sc \pmod{\mathbf{r}}$ (g and I are public), only if Alice knows s she can respond correctly to Bob with non-negligible probability (the probability that Alice correctly 'guesses' s in each round is $1/\mathbf{r}$, where one expects $\mathbf{r}$ to be of similar size to q), otherwise, if $s' \neq s$, in the verification step Bob will get $g^{r+c(s'-s)} \neq g^r \neq X$ and will reject the proof. After l runs of the protocol, Bob is convinced that Alice knows s but no other information about Alice leaks to Bob.

Therefore, the Schnorr identification is an example of a zero knowledge proof of the Discrete Log Problem (DLP). Note that Alice convinces Bob that she knows the discrete log of her public key I in respect to q without revealing any information to Bob about the discrete log s.

We saw in Section 2.8 that the security of a digital signature relies on the hardness of some mathematical problem. We can use the zero knowledge proof of discrete log, described above, to produce signatures assuming only that the DLP is hard. The trick is to replace interaction from the identification protocol with the output of a hash function h, which can be represented by a random oracle for a formal proof.

In the Schnorr signature, if Alice wants to sign a message m, she chooses a random r, computes $u = g^r \pmod q$, $c = h(m||u)$ and $y = r + sc \pmod{\mathbf{r}}$. Then, Alice's signature of m is (y, c). Notice that the interactive challenge was replaced with a hash value. This is known as "applying the Fiat-Shamir transform".

To verify Alice's signature, Bob (after checking if the public information is authentic) computes $z = g^y I^c \pmod q$ and $c' = h(m||z)$. He accepts Alice's signature if and only if $c = c'$.

Chapter 3

Computing Discrete Logs on Elliptic Curves

Contents

This chapter continues presenting background topics. We present methods to solve the Discrete Log Problem on Elliptic Curves, with emphasis on low-memory methods, since such methods are adapted to solve related problems later in this thesis.

We begin the chapter defining the elliptic curve discrete log problem (ECDLP), then we briefly present a time/memory tradeoff algorithm to solve it, namely, the baby-step-giant-step algorithm. In the main part of this chapter, we present and discuss low-memory algorithms to solve the ECDLP.

3.1 Elliptic Curve Discrete Log Problem (ECDLP)

Definition 8. *Let q be a prime power and $E/\mathbb{F}_q$ an elliptic curve defined over the finite field $\mathbb{F}_q$. We define $E(\mathbb{F}_q)[\mathbf{r}]$ to be $\{P \in E(\mathbb{F}_q) : [\mathbf{r}]P = \mathcal{O}\}$. In other words, it is "the group of points of order $\mathbf{r}$ in an elliptic curve E over $\mathbb{F}_q$".*

Definition 9. *Let $\mathbf{r} \mid \#E(\mathbb{F}_q)$ be a prime such that $E(\mathbb{F}_q)[\mathbf{r}]$ is cyclic. The* elliptic curve discrete logarithm problem *(ECDLP) is: given $P, Q \in E(\mathbb{F}_q)[\mathbf{r}]$, find $n \in \mathbb{Z}/\mathbf{r}\mathbb{Z}$ such that $Q = [n]P$.*

This is a fundamental computational problem with applications in elliptic curve public key cryptography (ECC).

The naive way to solve the ECDLP is by using exhaustive search over all possible values $n \leqslant \mathbf{r}$. In other words, this naive approach takes $O(\mathbf{r})$ steps (if $\mathbf{r}$ has k bits, the running time is $O(2^k)$ and therefore, exponential in k). However, it is well-known that one can compute n in $O(\sqrt{\mathbf{r}}) = O(2^{k/2})$ group operations using Shanks' baby-step-giant-step algorithm [16], or Pollard's [41, 42] low-memory randomised algorithms. As remarked by Galbraith [23]: "due to the Pohlig-Hellman algorithm (which reduces the problem to subgroups of prime order) we always restrict to the case where the point P has large prime order. Then the only algorithms which are applicable for all elliptic curves are the methods of Shanks and Pollard, and these methods have exponential complexity.".

Definition 10. *Let N be the number of group operations one needs to perform to solve a computational problem using exhaustive search. According to E. Teske [55] we call an algorithm a* ***square-root algorithm*** *if it finds the solution in (expected) $\sqrt{N}$ group operations. See also [2, Chapter 19].*

Now we review the main details of BSGS and Pollard methods. See also [16, 37, 49, 52, 53, 54, 55] for further details.

3.2 Shanks' Baby-Step Giant-Step Method

Let P have prime order $\mathbf{r}$. Let $m = \lceil \sqrt{\mathbf{r}}\ \rceil$. There exists integers i and j, where $0 \leqslant i, j < m$, such that $n = i + jm$. So, $Q = [i + jm]P$.

Basically, Shanks' algorithm precomputes a table T ("baby steps") which consists of pairs $(i, Q - [i]P)$ for $0 \leqslant i < m$. Notice that T has size $\sim \sqrt{\mathbf{r}}$. After sorting all values in T by the second component (i.e. $Q - [i]P$), we compute the "giant steps", which consist of pairs $(j, [jm]P)$ for $0 \leqslant j < m$, and check if $[jm]P$ is equal to any of the second components of T. When a match is found, we have $Q - [i]P = [jm]P$, so we get the corresponding value i and the discrete log is solved, since $n = i + jm$.

Now we analyse the running time of Shanks' algorithm. We may assume that logarithmic factors are negligible; the construction of T (baby steps) takes $O(\sqrt{\mathbf{r}})$ group operations; sorting elements in T by its second coordinate can be done efficiently in $O(\sqrt{\mathbf{r}}(\lg \mathbf{r}))$ bit operations (see, for example, [17] for sorting algorithms); the giant steps compute $O(\sqrt{\mathbf{r}})$ group operations and finally, searching $[jm]P$ in T can be done in logarithmic time, since T is *sorted* by its second component (see also [17] for searching algorithms).

Therefore, the running time is $\tilde{O}(\sqrt{\mathbf{r}})$. Note that we still have an exponential running time, but now the exponent is about one half of the exhaustive search case, i.e. $O(\sqrt{\mathbf{r}}) = O(2^{k/2})$. However, in this method the memory required to store T is $O(\sqrt{\mathbf{r}}(\lg \mathbf{r}))$ bits, which becomes prohibitive due to memory restrictions. This method is known as a *time/memory trade-off* algorithm.

3.3 Pollard-rho Method

We have seen that the BSGS method has running time bounded by $\tilde{O}(\sqrt{\mathbf{r}})$ and it needs to store $\tilde{O}(\sqrt{\mathbf{r}})$ elements, which quickly restricts its use. However, a natural question arises: "*Is it possible to keep the square root running time but with a much smaller storage requirement* ?". The answer to this question is "*yes, it is!*". We accomplish this using Pollard's algorithms [41, 42], but at the cost of an *expected* square root running time instead of an absolute bound.

3.3.1 The Birthday Paradox

Before discussing the Pollard-rho method, we revise a concept which will be useful in the analysis of Pollard-rho and Pollard-kangaroo algorithms. The reason for the name "Birthday Paradox" comes from the surprisingly large probability of two people sharing the same birthday in a room with only 23 people. At the end of this section we calculate this probability.

Consider the following example: let us say we have m balls in a box, each one with a different colour. We pick up one ball at a time, write down its colour and replace the ball in the box. Then we continue picking up balls until we get a match, i.e., a colour that was already written down.

We now prove the probability of obtaining at least one repeated colour and the expected number of balls that have to be taken out of the box before a repetition, following [58]:

Lemma 3.3.1. *In a box of m balls, each of them of a different colour, the probability, after k balls have been taken out of the box with replacement, that we have obtained at least one matching colour (or coincidence) is*

$$\boldsymbol{Pr} \approx 1 - e^{-k^2/2m}.$$

Proof. We first determine the probability of choosing k balls which all have different colours. We have m possible colours for the first ball, $m-1$ possible colours for the second ball and so on. For the k^{th} ball we have $(m-k+1)$ possible colours. We define:

$$m^{(k)} = m \cdot (m-1) \cdot (m-2) \cdots (m-k+1).$$

Therefore, the probability, after k balls have been taken out of the box, that a repeated colour *will not* occur is

$$\overline{\mathbf{Pr}} = \frac{m^{(k)}}{m^k} = (\tfrac{m}{m})(\tfrac{m-1}{m})(\tfrac{m-2}{m})\dots(\tfrac{m-(k-1)}{m}) = (1-\tfrac{1}{m})(1-\tfrac{2}{m})\dots(1-\tfrac{k-1}{m}).$$

which we can rewrite as

$$\overline{\mathbf{Pr}} = \prod_{i=1}^{k-1}(1 - \frac{i}{m}) \tag{3.1}$$

By definition, the Taylor series expansion of the exponential function e^{-x} is

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \dots$$

which can be approximated to

$$e^{-x} \approx 1 - x$$

when x is small.

Assuming m large, $1/m$ in Equation (3.1) is small, so, Equation (3.1) becomes:

$$\overline{\mathbf{Pr}} \approx \prod_{i=1}^{k-1} e^{-i/m} = e^{-\sum_{i=1}^{k-1} \frac{i}{m}} = e^{-\frac{k(k-1)}{2m}}.$$

And finally, we approximate to:

$$\overline{\mathbf{Pr}} \approx e^{-k^2/2m}$$

Therefore, the probability that a collision *will* occur in the first k steps is the complement of $\overline{\mathbf{Pr}}$:

$$\mathbf{Pr} \approx 1 - e^{-k^2/2m}$$

$\square$

And now we state a Theorem:

Theorem 3.3.2. *As m becomes larger the expected number of balls we have* *to take out of the box before we obtain the first collision is asymptotic to*

$$\sqrt{\frac{\pi m}{2}}.$$

Proof. Let X be the random variable for the number of elements of a set of size m that must be selected at random with replacement before any element is selected twice. Then, the expectation of X is

$$E(X) = \sum_{k=1}^{\infty} k.\mathbf{Pr}(X = k) = \underbrace{\sum_{k=1}^{\infty} k.\Big(\mathbf{Pr}(X > k-1) - \mathbf{Pr}(X > k)\Big)}_{(*)}.$$

Expanding $(*)$, we have:

$$1 \cdot (\mathbf{Pr}(X > 0) - \mathbf{Pr}(X > 1)) + 2 \cdot (\mathbf{Pr}(X > 1) - \mathbf{Pr}(X > 2)) + \ldots$$

which is equal to

$$\mathbf{Pr}(X > 0) + \mathbf{Pr}(X > 1) + \mathbf{Pr}(X > 2) + \ldots$$

and therefore:

$$E(X) = \sum_{k=0}^{\infty} \mathbf{Pr}(X > k).$$

From Lemma 3.3.1:

$$\mathbf{Pr}(X \leqslant k) = (1-1/m)(1-2/m)(1-3/m)\ldots(1-(k-1)/m) \approx 1 - e^{-k^2/2m}.$$

Hence

$$E(X) \approx \sum_{k=0}^{\infty} e^{-k^2/2m} \approx \int_0^{\infty} e^{-k^2/2m} dk = \sqrt{\pi m/2}.$$

According to [58], since $e^{-k^2/2m}$ is a monotonically decreasing function and its maximum value is 1, the error in approximating the sum with the integral is at most 1, which completes the proof. $\square$

Now, back to the room with 23 people, the probability of at least two people having the same birthday is $1 - \frac{365^{(23)}}{365^{23}} \approx 1 - e^{-(23)^2/2(365)}$, which exceeds 1/2.

3.3.2 The Idea of Pollard-rho

Now we outline the Pollard-rho method to find discrete logs. We recall that our problem is: given $P, Q \in E(\mathbb{F}_q)$, find $n \in \mathbb{Z}/\mathbf{r}\mathbb{Z}$ such that $Q = [n]P$. Roughly speaking, the basic idea of Pollard is to define a pseudorandom walk on the cyclic group. The sequence will look like a "rho" (i.e., it will have a tail followed by a cycle). The next step is to find a collision and when the collision is found we have the DLP solved.

Now we give the details of each phase.

3.3.2.1 A Pseudorandom Walk

For Pollard methods, it is essential to define a walk $(R_0, R_1, \ldots)$ according to a deterministic pseudorandom rule. The trick to simulate a pseudorandom walk is to choose each step of the walk as a function of the current position. We partition the group into s sets and define the walk in a piecewise way. As a result, we construct a deterministic walk which behaves like a pseudorandom walk.

There are a number of ways to compute an index i from an elliptic curve point R. For example, i can be the x-coordinate of R reduced modulo s.

Definition 11. *Let R be a point in the walk. Let $E(\mathbb{F}_q)$ be a cyclic group and let s be the number of disjoint subsets in which $E(\mathbb{F}_q)$ is divided. Let x_R be the x-coordinate of R. We define a function $\phi : E(\mathbb{F}_q) \to \mathbb{N}$ as follows:*

$$\phi(R) = x_R \bmod s \ .$$

Let $s = 3$, $S_i = \{R \in E(\mathbb{F}_q) : \phi(R) = i\}$ for $i = 0, 1, 2$. We assume that S_0, S_1 and S_2 have approximately the same size.

Definition 12. *Given a point R_0 and $a_0, b_0 \in \mathbb{Z}$, such that $R_0 = [a_0]P + [b_0]Q$, we define a pseudorandom walk $(R_0, R_1, \ldots)$ as follows:*

$$(R_{i+1}, a_{i+1}, b_{i+1}) = f(R_i, a_i, b_i) = \begin{cases} (R_i + P, a_i + 1, b_i) & \textit{if } \phi(R_i) = 0 \\ (R_i + R_i, 2a_i, 2b_i) & \textit{if } \phi(R_i) = 1 \\ (R_i + Q, a_i, b_i + 1) & \textit{if } \phi(R_i) = 2 \end{cases}$$

We stress that all a_{i+1} and b_{i+1} are computed modulo $\mathbf{r}$. For each triple (R_i, a_i, b_i), we have $R_i = [a_i]P + [b_i]Q$. In the serial case, we start our pseudorandom walk with $R_0 = P$, $a_0 = 1$ and $b_0 = 0$.

3.3.2.2 The Rho-shape of the Pseudorandom Walk

Since $E(\mathbb{F}_q)$ is finite, eventually we will get a collision, i.e., a pair of indices $1 \leqslant i < j$ such that $R_i = R_j$. We assume j is the smallest integer with this property. So,

$$R_{i+1} = f(R_i) = f(R_j) = R_{j+1}$$

and the sequence becomes cyclic. If we "draw" the pseudorandom sequence, it is easy to see a *tail* followed by a *cycle*. In other words, it will look like the Greek letter *rho*, i.e.

$$\rho \,.$$

Now, our goal is to find a collision. According to [49], in such sequences, the tail and the cycle have expected length $\sqrt{\pi \mathbf{r}/8}$. However, it might happen that the length of the cycle is much longer, so it will take more time for the first collision.

If we store every R_i and try to find the *first* index j such that $R_j = R_i$ for some i (which means that the "tail" finished and the "cycle" is beginning) we will need a large storage space (i.e., $\approx \sqrt{\mathbf{r}}$, since the expected tail length is $\sqrt{\pi \mathbf{r}/8}$). In other words, this would give no advantage over the BSGS method.

3.3.2.3 Finding Cycles

Since after the first collision the sequence becomes cyclic, we can use any cycle finding method, for example, Floyd's method or Brent's method [11].

We remark that R. Brent [11] claims that, on average, his cycle finding algorithm runs around 36% more quickly than Floyd's and that it speeds up the Pollard-rho algorithm by around 24%. We also remark that Sedgewick, Szymanski, and Yao [47] showed that a collision can be found approximately three times earlier if we store a small number of values from the R_i sequence. That approach requires that we have memory space greater than $O(1)$, but, certainly, much smaller than $O(\sqrt{\mathbf{r}})$.

Despite being less efficient, Floyd's method is simpler to understand, and since in real life we use parallelised versions where finding cycles is not necessary, we give the details of Floyd's method only: given the pair (x_i, x_{2i}) we compute $(x_{i+1}, x_{2i+2}) = \big(f(x_i), f(f(x_{2i}))\big)$ and stop when we find $x_m = x_{2m}$.

It is easy to check that $x_m = x_{2m}$ will occur for some $m < j$. Let x_{i-1} be the last point in the tail, before the cycle begins. Hence x_i is the first point in the cycle. If x_{j-1} is the point *in the cycle* that immediately precedes x_i, then $x_j = x_i$ (i.e., the first collision occurred), which means that the cycle has length $l = j - i$ (see Figure 3.1). Let $k = \lceil i/l \rceil$. Any point x_m in the cycle coincides with $x_{(m+kl)}$. Since the cycle is endless, when $m = kl$ we have $x_{kl} = x_{(2kl)}$. Note that $m < (i/l + 1)l = i + l = j$.

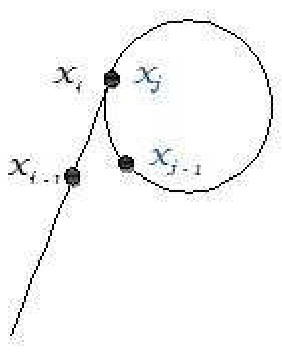

Figure 3.1: Here we can see a tail and a cycle in a rho-shape walk.

Using Floyd's method, the required memory is $O(1)$ group elements, therefore, this is a *low memory* method.

3.3.2.4 Solving the ECDLP

In every step of the walk we compute triples (R_i, a_i, b_i) and (R_{2i}, a_{2i}, b_{2i}) and check if $R_i = R_{2i}$. Notice that at each "step" we compute three group operations (see lines 23, 24, 25 in Algorithm 1). When a collision is found, we have $[a_i]P + [b_i]Q = [a_{2i}]P + [b_{2i}]Q$ and we can compute the discrete log (as long as $(b_{2i} - b_i) \not\equiv 0 \bmod \mathbf{r}$) as follows:

$$n = (a_i - a_{2i})(b_{2i} - b_i)^{-1} \bmod \mathbf{r} \tag{3.2}$$

Since $\mathbf{r}$ is large, we can assume that the probability that $b_{2i} \equiv b_i \bmod \mathbf{r}$ is small enough to be ignored.

Note that in the Pollard-rho method, the probability of success is well defined (i.e., a collision *will* occur if the sequence is sufficiently large), but the running time is expected, rather than absolutely bounded.

3.3.3 Analysis

In this section we summarise the Pollard-rho method more formally, presenting the algorithm and the running time analysis.

3.3.3.1 Pollard-rho Algorithm

Algorithm 1 presents the Pollard-rho method described above. Note that we needed to assume that f behaves like a truly random function for the computing implementation. In the analysis, however, we define f as a random function from $\langle P \rangle \times \mathbb{Z}/\mathbf{r}\mathbb{Z} \times \mathbb{Z}/\mathbf{r}\mathbb{Z}$ to itself.

Algorithm 1 Pollard-rho discrete log

```
Input: group $E(\mathbb{F}_q)$; points $P, Q \in E(\mathbb{F}_q)$; integers $a_0, b_0$; point $R_0 = [a_0]P + [b_0]Q$
Output: integer $n$, such that $Q = [n]P$
function $\phi(R)$
 $\phi = x_R \bmod 3$
 Return $\phi$
end function

function $f(R, a, b)$
if $\phi(R) = 0$ then
    $f \leftarrow (R + P, a + 1, b)$
else
    if $\phi(R) = 1$ then
        $f \leftarrow ([2]R, 2a, 2b)$
    else
        $f \leftarrow (R + Q, a, b + 1)$
    end if
end if
Return $f$
end function

main
$(R, a, b) \leftarrow f(P, 1, 0)$
$(R', a', b') \leftarrow f(R, a, b)$
while $R \neq R'$ do
    $(R, a, b) \leftarrow f(R, a, b)$
    $(R', a', b') \leftarrow f(R', a', b')$
    $(R', a', b') \leftarrow f(R', a', b')$
end while
if $(b' - b) \equiv 0 \pmod{\mathbf{r}}$ then
    Output "failure"
else
    Output $n = (a - a')(b' - b)^{-1} \pmod{\mathbf{r}}$
end if
end main
```

3.3.3.2 Running time

Notice that Algorithm 1 will either output "failure" or "n" (the solution for the DLP) after a while-loop. We use the Birthday Paradox to conclude that the expected running time of the Pollard-rho algorithm is $\tilde{O}(\sqrt{\mathbf{r}})$.

Theorem 3.3.3. *Suppose the pseudorandom walk in Algorithm 1 is replaced by a truly random function. The expected number of group operations in Algorithm 1 using Floyd's cycle finding method is at most* $3\sqrt{\pi\mathbf{r}/2}$.

Proof. From the Birthday Paradox (Theorem 3.3.2), the expected length of the rho is $\sqrt{\pi\mathbf{r}/2}$. Using Floyd's cycle finding method, in each "step" we compute 3 group operations. The number of steps is at most the length of the rho. Therefore, the expected number of group operations in Pollard-rho algorithm using Floyd's cycle finding method is at most $3\sqrt{\pi\mathbf{r}/2}$. □

3.4 Pollard-kangaroo Method

This method is a variant of the previous one when we know in advance that n lies in a short interval[1]. In other words, we have $n \in [a, b]$ for known a, b and $w = b - a$ (i.e., w is the length of the interval). Our problem remains the same: let $E(\mathbb{F}_q)$ be a cyclic group of order $\mathbf{r}$, given $P, Q \in E(\mathbb{F}_q)$, find $n \in [a, b]$ such that $Q = [n]P$.

3.4.1 The Idea of Pollard-kangaroo

Here follows the basic idea, according to Pollard [41]: "we want to catch a [wild] kangaroo W which is travelling along a known path in a series of what appear to be unpredictable bounds; in reality, their lengths are a function of the state of the ground at the point of take-off. We suppose

[1]See Section 3.5 for an analysis of Pollard-kangaroo when $w = \mathbf{r}$.

that this function takes values at random from an integer set S, of mean α and largest member L. We require the services of a second tame kangaroo T of identical jumping behaviour. We cause T to start at some point T_0 and take N bounds, arriving at T_N. A hole dug at this point, and suitably camouflaged, will catch the wild kangaroo W *if* he lands on any points $T_0, T_1, \ldots, T_N$".

Now, unlike Pollard-rho, it is possible that the wild kangaroo *never* touches any of the points followed by the tame kangaroo. In this case, W will not fall in the trap and will not be caught. Hence, in the Pollard-kangaroo method, the running time is well defined, but the probability of collision is expected, rather than absolutely bounded.

The Pollard-kangaroo method is also known as the "Pollard-lambda method" (actually, its author prefers the former [42], because the parallel version of Pollard-rho has a lambda-shape, rather than a rho-shape). Now unlike Pollard-rho, we use *two* pseudorandom walks instead of one. If there is a collision, the pseudorandom sequences will look like the Greek letter *lambda*, i.e.

$$\lambda \ .$$

Now we outline the kangaroo method following [49]. Basically, we define a pseudorandom walk, then we discuss how to compute the DLP if the wild kangaroo lands in any of the footprints of the tame kangaroo.

3.4.1.1 The Pseudorandom Walks

Let $E(\mathbb{F}_q)$ be a cyclic group of order $\mathbf{r}$. Let s be the number of partitions of the group and let $S = \{s_0, s_1, \ldots, s_{s-1}\}$ be a set of non-decreasing integers. Let α be the mean of the elements of the set S, let β be a parameter to be defined later and let integers $N = \alpha\beta$ and M be respectively the number of steps taken by the tame and the wild kangaroos. We can view each element of the set S as the length of a "jump". We define a pseudorandom walk using a similar principle as in Section 3.3.2.1. We recall that $\phi(R)$ maps the x-coordinate of a point R to an integer modulo s.

Tame kangaroo: T starts at some known point $T_0 = [c_0]P$, where c_0 is to be defined later. He jumps according to the following pseudorandom walk:

$$T_{i+1} = T_i + [s_{\phi(T_i)}]P \text{ for } i = 0, \ldots, N \ ,$$

which is efficiently calculated if the values $[s_i]P$ are precomputed. We also set $c_{i+1} = c_i + s_{\phi(T_i)}$, so that $T_i = [c_i]P$. Notice that $c_i - c_0$ is the distance travelled. We then store T_N. Notice that at this point, we know the discrete logarithm of T_N with respect to P, i.e., $T_N = [c_N]P$.

Wild kangaroo: W starts at Q and jumps according to the following pseudorandom walk:

$$W_{i+1} = W_i + [s_{\phi(W_i)}]P \text{ for } i = 0, \ldots, M \ .$$

We also set $d_0 = 0$ and $d_{i+1} = d_i + s_{\phi(W_i)}$, where d_i is the distance travelled from $Q = [n]P$ to W_i. Notice that we have $W_i = [n + d_i]P$.

3.4.1.2 Solving the ECDLP

If the wild kangaroo W at some point meets the path followed by the tame kangaroo T, then W will continue in the path originally followed by T until he reaches $W_m = T_N$ for some $m \leqslant M$. Hence:

$$[c_N]P = T_N = W_m = [n + d_m]P \ .$$

Thus, we have the solution to the DLP, since

$$n = c_N - d_m \ .$$

If the distance travelled by W (i.e., $d_m \approx M\alpha$) exceeds $c_N - a$ we can stop W, because he passed the trap.

In case we do not find a collision we can start another wild kangaroo at a new point, e.g., $Q' = Q + [z]P$, for a known integer z.

3.4.2 Choice of Parameters

- Set S - It is usual to choose s_i as powers of two, such that the mean step size is $g\sqrt{w}$, for some constant g (according to [42], this a good choice, but he does not claim that it is the *best* choice).
- s - If $s_i = 2^i$ for $0 \leqslant i < s$, the mean α is $\frac{2^s-1}{s}$, so we can choose $s \approx \frac{1}{2}\log_2(w)$.
- Starting point of tame kangaroo (T_0) - N. Smart [49] and many authors choose $[b]P$ as the starting point, but other choices are possible, for

instance, $[a]P$ or $[(a+b)/2]P$. Taking $T_0 = [b]P$ makes $c_0 = b$;

- M and N - We will prove below that a collision is expected after approximately $\sqrt{w}$ steps (remember that w is the length of the interval in which the discrete logarithm n is known to lie), so we set the number of steps of the tame kangaroo to be $N \approx \lceil \sqrt{w}\, \rceil$. If we set the starting point of T as $[b]P$, clearly we need $M > N$, so we choose $M = \gamma N$ for some positive constant γ to be defined later. The precise choice for γ will be presented in Section 3.4.4.

The number of steps is at most $N + M = (1+\gamma)\lceil \sqrt{w}\, \rceil$, so the running time is $\tilde{O}(\sqrt{w})$. As we only store the final position of the tame kangaroo (T_N), the memory required is $O(1)$ group elements so this is also a low memory method.

Figure 3.2 is a sketch of a successful collision, using parameters discussed in this subsection.

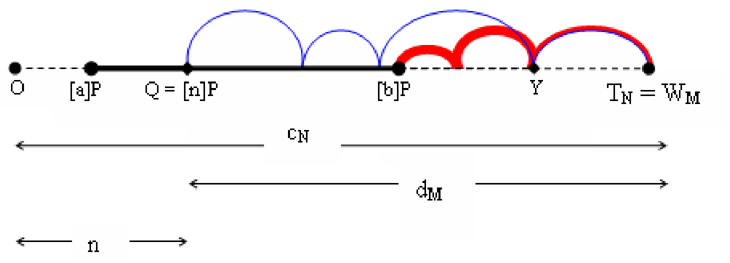

Figure 3.2: Tame kangaroo T starts at $[b]P$ and takes N jumps, stopping at T_N. At this point, the distance travelled by T is c_N (from the origin O). Wild kangaroo starts at Q and if he meets the path followed by T at some Y, he continues the same path and finally lands at the trap $T_N = W_M$. The distance travelled by W is $n + d_M$.

3.4.3 Analysis

Now, like we have done with Pollard-rho, we present a formal analysis of Pollard-kangaroo.

3.4.3.1 Algorithm

Algorithm 2 presents the Pollard-kangaroo method described above.

3.4.3.2 Probability of Success

Now we compute the probability that the wild kangaroo will be caught (i.e., the probability of successfully solving the DLP), according to [58].

Lemma 3.4.1. *Let M and N be the number of steps taken respectively by the wild and the tame kangaroos. Let α be the mean of the pseudorandom steps in set S and let β be an integer such that $N = \alpha\beta$. Let $\boldsymbol{Pr}_{[\lambda]}$ be the probability that the wild kangaroo, after passing $[b]P$ and taking about $\alpha\beta$ jumps from $[b]P$, will meet the path followed by the tame kangaroo and therefore, be caught in the trap. Under the assumption that the pseudorandom walk behaves the same as a random walk, then*

$$\boldsymbol{Pr}_{[\lambda]} \approx 1 - (1 - 1/\alpha)^{\alpha\beta} \approx 1 - e^{-\beta}$$

Proof. Since the mean of the steps in set S is α, once the wild kangaroo has passed the starting point of the tame kangaroo, each jump of the wild kangaroo has a heuristically independent probability of about $1/\alpha$ of touching a point already followed by the tame kangaroo. Since the number of

Algorithm 2 Pollard-kangaroo discrete log

```
Input: group $E(\mathbb{F}_q)$, points $P, Q \in E(\mathbb{F}_q)$, integers $a, b$
Output: integer $n$ such that $a \leqslant n \leqslant b$ and $Q = [n]P$.
 1: Choose number of partitions $s$
 2: Choose starting point of tame kangaroo $T_0 = [c_0]P$
 3: Choose steps $s_i$    /* Usually, $s_i = 2^i$ */
 4: Choose walk lengths $M, N \in \mathbb{N}\ \mathcal{S}_j$, $0 \leq j < s$.
 5: function $\phi(R)$
 6:  $\phi = x_R \bmod s$
 7:  Return $\phi$
 8: end function
 9:
10: Precompute $S_i = [s_i]P$ for $0 \leq i < s$
11: [Tame Kangaroo]
12: $T \leftarrow T_0$
13: $c \leftarrow c_0$
14: for $i = 1$ to $N$ do
15:    Compute $j = \phi(T)$
16:    $T \leftarrow T + S_j$
17:    $c = c + s_j$
18: end for
19: [Wild Kangaroo]
20: $W \leftarrow Q$
21: $d \leftarrow 0$
22: $i \leftarrow 0$
23: repeat
24:    Compute $j' = \phi(W)$
25:    $W \leftarrow W + S_{j'}$
26:    $d \leftarrow d + s_{j'}$
27:    $i \leftarrow i + 1$
28: until $(T = W)$ or $(i > M)$
29: if $T = W$ then
30:    Output $n = (c - d)$
31: else
32:    Output "failure"
33: end if
```

steps taken by T is $N = \alpha\beta$, then W has approximately $(1-1/\alpha)^{\alpha\beta}$ chances of avoiding the footprints of T. Therefore, the probability of success is $\mathbf{Pr}_{[\lambda]} \approx 1 - (1 - 1/\alpha)^{\alpha\beta} \approx 1 - e^{-\beta}$ □

Notice that for $\beta = 1, 2, 3, 4$ the probability of success is respectively $0.63, 0.86, 0.95, 0.98$, which means that the size of N defines the probability of success.

3.4.3.3 Running Time

We also follow [58] to prove the running time of Pollard-kangaroo.

Theorem 3.4.2. *Consider β as defined in Lemma 3.4.1. The expected number of group operations in Pollard-kangaroo algorithm in the average case is $2\sqrt{\beta w}$ and in the worst case is $2\sqrt{2\beta w}$.*

Proof. It is easy to see that the number of group operations in Algorithm 2 is $Set + N + M$, where Set represents the construction of set S (i.e., computing $S_i = [s_i]P$ for $0 \leqslant i < s$). Assuming that Set can be precomputed, we can simplify the running time to be $N + M$.

We recall that $n \in [a, b]$ for known a, b and $w = b - a$ (i.e., w is the length of the interval). The tame kangaroo T starts at b and jumps $N = \alpha\beta$ steps. Since the mean of set S, which represents each step, is α, T stops at average distance $w + \alpha^2\beta$ from a.

The wild kangaroo W starts at n (unknown) and after it passes b, W takes about $\alpha\beta$ jumps to reach T. Since the average starting point of W is at $w/2$, the expected number of steps of W in the average case is $w/(2\alpha) + \alpha\beta$.

Therefore, the total running time is approximately

$$N + M = \alpha\beta + w/(2\alpha) + \alpha\beta = 2\alpha\beta + w/(2\alpha).$$

If we choose S such that its mean α is $g\lceil\sqrt{w}\rceil$ for some positive constant g and set $N = \alpha\beta$, we get a total of

$$\left(2g\beta\sqrt{w} + 1/2g\right)\sqrt{w} = \left(\frac{4g^2\beta + 1}{2g}\right)\sqrt{w} \tag{3.3}$$

group operations.

Now we want to minimise $f(x) = \frac{4x^2\beta+1}{2x}$. We get $f'(x) = \frac{(8x\beta).2x-(4x^2\beta+1).2}{4x^2}$. Taking $f'(x) = 0 \Rightarrow 8x^2\beta - 2 = 0 \therefore x^2 = \frac{1}{4\beta} \Rightarrow g = \frac{1}{2\sqrt{\beta}}$.

Replacing the value of g in Equation (3.3), it follows that the number of group operations in the average case is $2\sqrt{\beta w}$.

In the worst case, the wild kangaroo will start in the beginning of the interval (i.e., $[a]P$), so the expected number of steps is $w/\alpha + \alpha\beta$ and the total running time is

$$N + M = \alpha\beta + w/\alpha + \alpha\beta = 2\alpha\beta + w/\alpha.$$

Taking $\alpha = g\lceil\sqrt{w}\rceil$, we get

$$2\beta g\sqrt{w} + w/g\sqrt{w} = \left(\frac{2g^2\beta + 1}{g}\right)\sqrt{w} \tag{3.4}$$

group operations.

Minimising $f(x) = \frac{2x^2\beta+1}{x}$, we get $f'(x) = \frac{(4x\beta).x-(2x^2\beta+1).1}{x}$. Taking $f'(x) = 0 \Rightarrow 2x^2\beta - 1 = 0 \therefore x^2 = \frac{1}{2\beta} \Rightarrow g = \frac{1}{\sqrt{2\beta}}$.

Replacing the value of g in Equation (3.4), it follows that the number of group operations in the average case is $2\sqrt{2\beta w}$. $\square$

3.4.4 Bounding the Running Time

Notice that the result of Theorem 3.4.2 gives a running time, which succeeds with probability $\approx 1 - e^{-\beta}$, as defined above. We remark that van Oorschot and Wiener [58] instead of giving a running time which outputs the solution with some probability, calculated the average running time in that the algorithm runs as many times as needed until a collision is found. At each new round of the algorithm, one uses another wild kangaroo, starting from $Q + [z]P$, for a known integer z. We sketch their approach:

1. Tame kangaroo takes $N = \alpha\beta$ steps;
2. From Lemma 3.4.1, the expected number of trials for a success to occur is $1/(1 - e^{-\beta})$. So, the wild kangaroo is expected to succeed once and to fail $1/(1 - e^{-\beta}) - 1$ times, which means that the total number of steps of all wild kangaroos necessary for a collision occur is $(w/\alpha + \alpha\beta)(1/(1 - e^{-\beta}) - 1) + w/(2\alpha) + \alpha\beta$.

So, the total running time is the number of steps of the tame kangaroo plus the total number of steps of all wild kangaroos

$$\underbrace{\alpha\beta}_{N} + (\alpha\beta + w/\alpha)/(1 - e^{-\beta}) - w/(2\alpha),$$

which is minimized when $\alpha = \sqrt{w(1 + e^{-\beta})/(2\beta(2 - e^{-\beta}))}$. According to [58], using numerical techniques, we get $\beta = 1.39$ and $\alpha = 0.51\sqrt{w}$, and therefore, $N = \alpha\beta \approx 0.7089\sqrt{w}$, $M \approx 2.57\sqrt{w} = \gamma N$, $\gamma \approx 3.62$ and the total running time is $3.28\sqrt{w}$ group operations.

Notice that if we use the value of β above ($\beta = 1.39$) in the result of Lemma 3.4.1, the probability of success is around 0.75.

3.5 Pollard-rho vs. Pollard-kangaroo

By studying the Pollard-rho and the Pollard-kangaroo methods, we see that there is a conceptual difference in terms of running time and probability of success. In the Pollard-rho method, the algorithm will *always* output a solution (unless $(b' - b) \equiv 0 \bmod \mathbf{r}$, which occurs with small probability if $\mathbf{r}$ is large) and we have an expected number of steps, and therefore, an expected running time. On the other hand, in the Pollard-kangaroo method, we have a fixed running time (i.e., a fixed number of steps) and the algorithm can either output the solution "n" or "failure" with some probability.

Usually, Pollard-kangaroo is more appropriate when we know in advance that the discrete log lies in some interval, $[a, b]$ (i.e., the solution n is such that $a \leqslant n \leqslant b$). However it is also possible to use Pollard-kangaroo when $w = \mathbf{r}$. Pollard [42] states that if $w = \mathbf{r}$, although the running time is asymptotically the same as in Pollard-rho method, the constants will be greater in the kangaroo method, thus the "rho" method is preferred. We proved that if Floyd's cycle finding method is used, the expected running time of Pollard-rho in a serial computer is around $3\sqrt{\pi \mathbf{r}/2} \approx 3.76\lceil\sqrt{w}\ \rceil$. Using the best known cycle finding algorithm for Pollard-rho (Brent's), we get an improvement of around 24% in the running time, thus $\approx 2.86\lceil\sqrt{w}\ \rceil$, which is better than the expected running time of Pollard-kangaroo on a serial computer, $3.28\lceil\sqrt{w}\ \rceil$ (and even better if we have enough memory space to use the idea of Sedgewick, Szymanski, and Yao [47]).

In practice, many processors are used in parallel to find a discrete log (see next section). In this case, van Oorschot and Wiener [58] showed that when $w = \mathbf{r}$ and parallel collision search is used, Pollard-rho is around 1.60 times faster than Pollard-kangaroo.

3.6 Parallelised Pollard Methods

In real life, we use many processors in parallel to try to solve a discrete logarithm problem. The naive way to parallelise Pollard-kangaroo methods is to divide the interval in which the discrete log lies (i.e., w) into $\mathbf{P}$ equal parts and use each of the $\mathbf{P}$ processors to search one part. The running time is $O((w/\mathbf{P})^{1/2})$.

However, both Pollard-rho and kangaroo methods can be parallelised achieving *linear* speedup. According to van Oorschot and Wiener [58] the running times are respectively $O(\mathbf{r}^{1/2}/\mathbf{P})$ and $O(w^{1/2}/\mathbf{P})$ with $\mathbf{P}$ processors.

The idea is to use each processor starting a deterministic random walk from a different starting point, but using the same function to compute the next point in the walk. When a collision is found by two processors (or even by the same processor) we have the discrete log solved. As usual, the challenge is to detect collisions. The solution is to store only points which satisfy some condition (otherwise, if all points were stored, the storage requirements would be of the order of $O(\sqrt{w})$ or $O(\sqrt{\mathbf{r}})$). Such points are called *distinguished* points. We now sketch this approach, called "Parallel Collision Search". The proofs can be found in [58].

We now define a distinguished point, according to [58]:

Definition 13. *"A* distinguished point *is one that has some easily checked property such as having a fixed number of leading zero bits. During the pseudorandom walk, points that satisfy the distinguishing property are stored. Repetition can be detected when a distinguished point is encountered a second time. The distinguishing property is selected so that enough distinguished points are encountered to limit the number of steps past the beginning of the repetition, but not so many that they cannot be stored".*

We stress that the distinguished points found by the processors are stored by a server who coordinates the solution of the problem.

It is possible that a random walk falls into a loop such that no distinguished point is reached. This problem can be overcome by setting a maximum walk length and abandoning any walk which has length greater than the maximum walk length. Let θ be the proportion of points which satisfy the distinguishing property. Let $Top = c/\theta$ for some positive constant c be the maximum walk length. We stress that, in the running time analysis, we can consider the average number of steps needed to reach a distinguished point as $1/\theta$ and not c/θ, since most of the time, the distinguished point will be reached in $1/\theta$ steps, so the very few cases when it takes at most c/θ steps to reach (or not) a distinguished point do not have much effect in the total running time.

Now we analyse the average running time, according to [58]. Let $\mathbf{P}$ be the number of processors used in the parallel search. Let Top be the maximum walk length to find a distinguished point and let T_ρ and T_λ be respectively the Pollard-rho and Pollard-kangaroo running time of each individual processor using parallel collision search. When processors hit a distinguished point, they send information to the server. In the rho method, after hitting a distinguished point, the processor starts a new walk from a random point. In the kangaroo method, the processor continues in the same walk. Note that in the kangaroo method a processor can reach many distinguished points during its walk. At some time there is a collision, in the sense that two processors visit the same point. The server does not detect this until it receives the same distinguished point twice. Hence, each processor is expected to take $1/\theta$ further steps after the collision has occurred.

First we analyse the Pollard-rho case. From the Birthday Paradox, a collision is expected after $\sqrt{\pi\mathbf{r}/2}$ steps have been made in total. Hence

$$T_\rho = \sqrt{\pi\mathbf{r}/2}/\mathbf{P} + 1/\theta.$$

We remark that in the parallel version of Pollard-rho there is no need to use cycle finding methods.

In the Pollard-kangaroo case, van Oorschot and Wiener [58] use the mean step size in their analysis. Let α be the mean step size. We launch $\mathbf{P}/2$ tame kangaroos around the middle of the interval and $\mathbf{P}/2$ wild kangaroos around Q at the same time and store the distinguished points that each kangaroo reaches. Initially, the groups of tame and the wild kangaroos are separated by some distance between 0 and $w/2$. The average separation is $w/4$ and each trailing kangaroo takes approximately $T_1 = w/(4\alpha)$ steps to cover this distance. After that, each trailing kangaroo covers a region where about $(\mathbf{P}/2)/\alpha$ points have been visited by leading kangaroos. The probability that one of the $\mathbf{P}/2$ trailing kangaroos hits one of the footprints of a leading kangaroo at each step is about $(\mathbf{P}/2)^2/\alpha$. Hence the expected number of steps for each kangaroo before a collision occurs is $T_2 = 4\alpha/\mathbf{P}^2$. The minimum of $T_1 + T_2$ is $2\sqrt{w}/\mathbf{P}$ when $\alpha = (\mathbf{P}/4)\sqrt{w}$. Now we add the further $1/\theta$ steps required to reach the next distinguished point after a collision, and the total running time for each processor is:

$$T_\lambda = 2\sqrt{w}/\mathbf{P} + 1/\theta$$

group operations.

When a processor reaches a distinguished point already stored by the server, say, point D_1, the server checks if D_1 was reached by kangaroos of the same type (i.e., Tame-Tame or Wild-Wild). If "yes", the processor is told to move the trailing kangaroo some random distance forward, so as not to follow the very same path of the leading kangaroo. If "not", a *good collision* was found and the server solves the discrete log. J. Pollard [42] presented a different version for the parallel algorithm of [58], in that collisions between kangaroos of the same type are not possible. We remark that [56] shows that the expected number of Tame-Tame or Wild-Wild collisions in this setting is at most 2, so they are not a serious problem.

We remark that van Oorschot and Wiener [58] compared their parallel version of Pollard-rho and Pollard-kangaroo and concluded that when $w = \mathbf{r}$ the kangaroo method is approximately 1.60 times slower.

Note that [58] does not give an analysis for the success probability of the Parallel Pollard-kangaroo algorithm. This is because the processors always keep running even if they reach a distinguished point. They are only supposed to stop when the server tells them to stop. Hence, whenever the algorithm terminates the result is correct.

3.7 Summary

In this chapter, we revised the elliptic curve discrete log problem and some methods to solve it.

We first presented a time/memory tradeoff algorithm, baby-step giant-step, which solves the discrete log in $\tilde{O}(\sqrt{\mathbf{r}})$ group operations, but needs the same amount in memory space. For large $\mathbf{r}$, the memory consumption shortly becomes prohibitive.

Then, we presented and analysed low-memory algorithms with a heuristic rather than an absolute running time of $\tilde{O}(\sqrt{\mathbf{r}})$. In other words, Pollard-rho and Pollard-kangaroo methods are *expected* to solve the DLP with certain probabilities of success.

Chapter 4

Koblitz Curves and Frobenius Expansions

Contents

In this chapter, we briefly review some properties of Koblitz elliptic curves and Frobenius Expansions and how we can perform point multiplication efficiently using Frobenius expansions. We will see also that the arithmetic of Frobenius expansions has a non-standard behaviour when we require that all coefficients be only in $\{-1, 0, 1\}$. In addition, we experimentally estimate the number of equivalence classes of Frobenius expansions.

We explore the property that a τ-adic can be mapped to $x + y\tau$ for integers x, y of certain size and we give a proof for the bounds of x and y. We plot the the distribution of τ-adics and τ-NAFs. From these pictures, we conjecture that we can shorten the bounds for x and y.

We finish the chapter by studying an algebraic computation required for our τ-adic version of GPS (Girault-Poupard-Stern) identification protocol. In order to avoid a statistical attack in the τ-GPS protocol, we propose a randomisation step in the algorithm used to add and multiply τ-adics. We also state two heuristics that will be used in the proof of security of τ-GPS.

4.1 Koblitz Curves

We open this chapter defining Koblitz curves as follows. For further details see [33, 34, 50, 51]:

Definition 14. *A* Koblitz curve *is an elliptic curve E defined over a small finite field $\mathbb{F}_q$ such that the group $E(\mathbb{F}_{q^m})$ is suitable for cryptography for some $m > 1$.*

Definition 15. *Let $f(x) \in \mathbb{Z}_q[x]$ be an irreducible polynomial of degree m, and consider the finite field $\mathbb{F}_{q^m} = \mathbb{Z}_q[x]/(f(x))$. We say that $f(x)$ is a* normal polynomial *if the set $\{x, x^q, x^{q^2}, \ldots, x^{q^{m-1}}\}$ forms a basis for $\mathbb{F}_{q^m}$ over $\mathbb{Z}_q$. Such a basis is called* normal basis.

The most popular choice for Koblitz curves is curves over $\mathbb{F}_2$ and so we give the details in this case only.

Definition 16. *Let $a \in \{0,1\}$. Let $E : y^2 + xy = x^3 + ax^2 + 1$, with $a \in \{0,1\}$ be an elliptic curve defined over $\mathbb{F}_2$. Denote by $\mathcal{O}$ the point at infinity. Let $E(\mathbb{F}_{2^m})$ be the group of $\mathbb{F}_{2^m}$-rational points on E. We assume that m is a prime and that $\#E(\mathbb{F}_{2^m})$ has a large prime factor. In such curves, if $P = (x,y) \in E_a(\mathbb{F}_{2^m})$, then $Q = (x^2, y^2)$ also belongs to the same curve. The 2-power* Frobenius map *is defined as*

$$\begin{aligned} \tau : \ & E(\mathbb{F}_{2^m}) && \to E(\mathbb{F}_{2^m}) \\ & (x,y) && \mapsto (x^2, y^2) \\ & \mathcal{O} && \mapsto \mathcal{O}. \end{aligned}$$

Let $\bar{\mathbb{F}}_2$ be the algebraic closure of $\mathbb{F}_2$, i.e., $\bar{\mathbb{F}}_2$ is a field which contains the fields $\mathbb{F}_{2^n}$ for any n. Let $t = (-1)^{1-a}$. Then τ satisfies the quadratic characteristic polynomial $\tau^2(P) - t\tau(P) + [2]P = \mathcal{O}$ for all points $P \in E(\bar{\mathbb{F}}_2)$ (see [33]). In other words,

$$[2]P = t\tau(P) - \tau(\tau(P)). \tag{4.1}$$

Let x_P and y_P be the m-bit long coordinates of P. Notice that the τ operation maps each coordinate of P to its square. If $\mathbb{F}_{2^m}$ is represented by a normal basis, squaring (i.e., the operation $\tau(P)$) can be implemented by shifting each bit of x_P and y_P one bit to the left. This shifting is circular, which means that the most significant bit shifted one-bit to the left becomes the least significant bit and so on. For example, if 1001 and 1101 respectively represents x_P and y_P in binary, when multiplying P by powers of τ we get: $\tau(P) = (0011, 1011)$; $\tau^2(P) = (0110, 0111)$; $\tau^3(P) = (1100, 1110)$; $\tau^4(P) = (1001, 1101)$.

Notice that when we multiply by τ^m, each m-bit of x_P and y_P rotates m bits to the left, returning to their original state, which means that:

$$\tau^m(P) = P \text{ for all } P \in E(\mathbb{F}_{2^m}). \tag{4.2}$$

4.2 Frobenius Expansions

Definition 17. *Let $\mathcal{C}$ be the coefficient set for the τ-adic expansions (we always assume $0 \in \mathcal{C}$). For $L \in \mathbb{N}$ we define the set of Frobenius expansions of length L to be*

$$\mathcal{T}_L = \Big\{ x_0 + x_1\tau + x_2\tau^2 + \cdots + x_{L-1}\tau^{L-1} \mid x_j \in \mathcal{C} \Big\}. \tag{4.3}$$

For $x \in \mathcal{T}_L$ let $i \leqslant L-1$ be the largest index such that $x_i \neq 0$. We say that x has degree *i, denoted* $\deg(x) = i$. *The number of nonzero coefficients x_j is called the* weight.

We emphasize that we treat elements of $\mathcal{T}_L$ as polynomials and not as endomorphisms. We interchangeably use the terminology Frobenius expansion *and* τ-adic expansion *for such polynomials.*

We stress that in this chapter we assume $\mathcal{C}$ to be $\{-1, 0, 1\}$. This is the most popular choice, since one can benefit from the fact that the cost of addition and subtraction of points on elliptic curve are roughly the same. As a result, one can efficiently perform scalar multiplication using a "τ-and-(add or subtract)" algorithm (see Section 4.2.3).

4.2.1 τ-adic NAF Expansions

The concept of τ-NAF was introduced by J. Solinas in [50].

Definition 18. *An element* $\sum_{i=0}^{L-1} x_i\tau^i$ *is called a* τ*-adic NAF (where NAF stands for Non-Adjacent Form) if*

$$x_i x_{i+1} = 0 \text{ for all } i.$$

We will call a τ*-adic NAF as* τ*-NAF for short.*

In other words, there are no consecutive terms which are both nonzero.

An important fact about τ-NAF expansions is that, unlike standard τ-adics, we have a *unique* τ-NAF representation (of degree less than m) for an integer [51]. See Section 4.2.2 for a better understanding of uniqueness.

4.2.1.1 Number of τ-NAF of Degree less than L

It will be useful later in this thesis to know the number of τ-NAF of length less than L. So we use the following theorem (first stated by [50, 51]):

Theorem 4.2.1. *Let* $L \in \mathbb{Z}$ *be the length of a* τ*-NAF expansions. Let* N_L *be the number of possible* τ*-NAF with degree less than* L*. The number of possible* τ*-adic NAF expansions of degree less than* L *is*

$$N_L = \frac{4}{3}2^L - \frac{1}{3}(-1)^L = \frac{4}{3}2^L + O(1) \tag{4.4}$$

Proof. For any τ-NAF expansion with degree less than L, either the coefficient of τ^{L-1} is equal to zero (in which case it is a τ-NAF of length $L-1$)

or the coefficient of τ^{L-1} is nonzero (in which case the coefficient of τ^{L-2} is zero and the remaining polynomial is a τ-NAF of length $L-2$). Hence

$$N_L = N_{L-1} + 2N_{L-2}.$$

Solving the recurrence relation (see for example [13]) gives $N_L = c_1\alpha_1^L + c_2\alpha_2^L$ for some constants $c_1, c_2 \in \mathbb{Q}$ where α_i are the roots of $x^2 - x - 2 = 0$, i.e., $\alpha_1 = 2$ and $\alpha_2 = -1$. Taking $N_0 = 1$ and $N_1 = 3$ leads to $c_1 = 4/3$ and $c_2 = -1/3$ and the proof is complete. $\square$

4.2.2 Equivalence Classes of τ-adic Expansions

Definition 19. *We call two Frobenius expansions a, b* equivalent *if $[a]P = [b]P$ for all $P \in E(\bar{\mathbb{F}}_2)$ and write $a \equiv b$.*

It is easy to note that Frobenius expansions are not *unique*. Let T_1 be a random Frobenius expansion. Let Z be the characteristic polynomial $2 - t\tau + \tau^2 = 0$ (see Section 4.1). If we compute $T_1 + Z$ we get a Frobenius expansion $T_2 \equiv T_1$. For example, $1 \equiv -1 + t\tau - \tau^2$.

We now give a different definition of equivalence.

Definition 20. *Two Frobenius expansions a' and b', such that $a' \equiv b' \pmod{\tau^m - 1}$, are* equivalent with respect to a point *P, which we call P-*equivalent *for short, if $[a']P = [b']P$ for any point P defined over $\mathbb{F}_{2^m}$.*

Note that this definition depends on the point P. From Equation (4.2), it follows that $\mathcal{O} = (\tau^m - 1)P$. Then, for some k we can write:

$$[a']P = [b']P + k(\tau^m - 1)P = [b']P + [k]\mathcal{O} = [b']P.$$

It is easy to check that all equivalent τ-adics are also P-equivalent, since the equivalence notion holds for all $P \in E(\bar{\mathbb{F}}_2)$. On the other hand, it is not true that all P-equivalent τ-adics are also equivalent. For instance, let P be any point defined over $\mathbb{F}_{2^m}$. Clearly, τ^m is P-equivalent to 1 but $\tau^m \not\equiv 1$.

The drawback of using a random τ-adic lies in the fact that τ-adic expansions are not unique, so when one chooses a random τ-adic, there are several equivalent τ-adic expansions.

There are typically many different τ-adic expansions of degree less than some bound in any given equivalence class. An heuristic formula given in [21] for the average number of τ-adic representations of length at most $L + 2$ of a τ-NAF of length L is the integer closest to $0.9786\left(\frac{3}{2}\right)^L$. If all equivalence classes had this many elements then there would be at least $3^L/(0.9786(3/2)^{L-2}) \approx (2.3) \cdot 2^L$ different equivalence classes.

Experimentally, we used MAGMA [10] to compute all τ-adics of length $\leqslant L$ over $E(\mathbb{F}_{2^m})$. For each τ-adic, it was easy to check if it is a τ-NAF. Furthermore, we computed $Q = [n(\tau)]P$ for some point P and stored the points Q. Hence, after generating all τ-adics (i.e., 3^L) we have not only the total number of τ-NAFs of length $\leqslant L$ (i.e., $\approx \frac{4}{3}2^L$, see Section 4.2.1) but also the number of P-equivalence classes that can be represented as a τ-adic of maximum length L. Another way to count the number of P-equivalence classes would be to convert each τ-adic expansion to an integer, using the *eigenvalue of Frobenius* (see Section 4.2.7) and store the integer. Our experiments (see Table 4.1) suggest that the number of P-equivalence classes of τ-adic expansions of length L is bounded by $c(2^L)$, for some positive constant c, or simply $\tilde{O}(2^L)$.

The graphic on fig. 4.1 is based on Table 4.1 and shows the logarithm in base 2 of number of τ-adics, number of τ-NAF and number of P-equivalence classes. One can check that the number of τ-NAF and the number of P-equivalence classes are represented by parallel lines.

Table 4.1: Number of τ-adics, τ-NAFs and P-equivalence classes

L	# of τ-adics (3^L)	# of τ-NAF ($\approx \frac{4}{3}2^L$)	# of distinct P-equivalence classes
3	27	11	23
4	81	21	53
5	243	43	119
6	729	85	257
7	2187	171	534
8	6561	341	1133
9	19683	683	2335
10	59049	1365	4777
11	177147	2731	9711
12	531441	5461	19653
13	1594323	10923	39655
14	4782969	21845	79809
15	14348907	43691	160359

We now give a short motivation which justifies our interest for Frobenius expansions.

4.2.3 Efficient Point Multiplication Using Frobenius Expansions

In many applications of elliptic curve cryptography we need to compute point multiplication. Let P be a point of order $\mathbf{r}$ of an elliptic curve defined over a cyclic group $\mathcal{G}$. The natural way to compute the point multiplication $[n]P$ is to use the "double-and-add" algorithm (see Algorithm 3 and also [54, pg 265–266] for details).

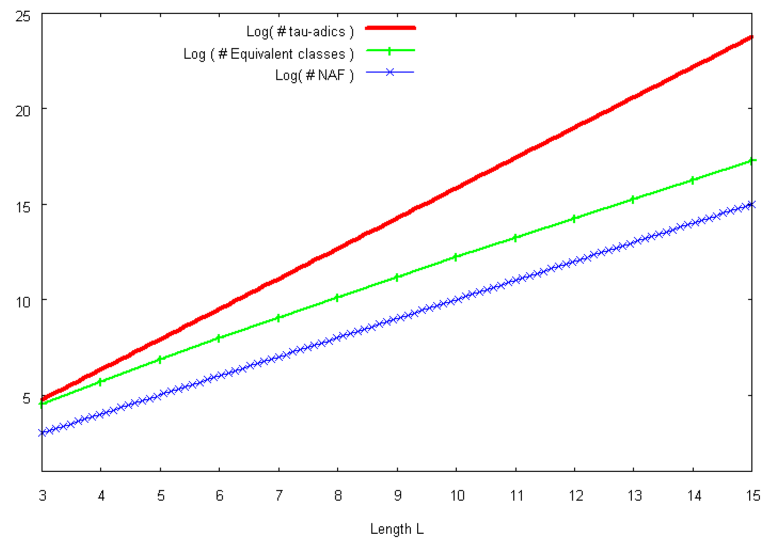

Figure 4.1: This graphic is based on the second, third and fourth columns of Table 4.1 and shows the logarithm in base 2 of each value.

If we work over a Koblitz curve, we have $\mathcal{G} = \mathbb{F}_{2^m}$ and the order $\mathbf{r}$ of P (and therefore, n as well) has size approximately 2^m. Basically, we start with a point Q being the point at infinity, write n in binary and move along each bit, from the left to the right (i.e., the first bit to be considered is the most significant bit of n). For each bit of n, we double Q, and if the bit is “1”, we add P. We remark that a variant called “double-and-(add or subtract)” algorithm can also be used when n is written in a binary signed representation (i.e., $n = b_m, b_{m-1}, \ldots, b_1, b_0$, for $b_i \in \{-1, 0, 1\}$). In that case, if the bit is “-1” we subtract P instead of adding it.

The total cost of $[n]P$ is m doublings plus w additions (or additions and subtractions if double-and-(add or subtract) is used), where w is the number of nonzero bits of n. We can simplify that, assuming that the cost of doublings and additions is roughly the same. So the total cost of $[n]P$ is

Algorithm 3 Double-and-Add Algorithm

```
Input: point $P, Q \in \mathcal{G}$; integer $n$
Output: point $Q = [n]P$
1: Compute the binary representation of $n$ ($n = b_m, b_{m-1}, \ldots, b_1, b_0$)
2: $Q \leftarrow \mathcal{O}$
3: for $i = m$ downto 0 do
4:     $Q \leftarrow 2Q$
5:     if $b_i = 1$ then
6:         $Q \leftarrow Q + P$
7:     end if
8: end for
9: Output $Q$
```

$m + w$ group operations.

However, when working over Koblitz curves, we can speed-up even more point multiplication using a trick proposed by N. Koblitz [33], in that we replace the integer n by its corresponding Frobenius expansion $n(\tau)$ (see also [36, 38, 50]):

$$n(\tau) = \sum_{j=0}^{L-1} n_j \tau^j$$

where the integers n_j lie in $\{-1, 0, 1\}$.

Since $E(\mathbb{F}_{2^m})[\mathbf{r}]$ is cyclic it follows that $\tau(P) = [\lambda]P$ for some $\lambda \in \mathbb{Z}/\mathbf{r}\mathbb{Z}$. If $n = \sum_{j=0}^{L-1} n_j \lambda^j \pmod{\mathbf{r}}$ then $Q = [n]P$ can be efficiently computed as

$$Q = [n(\tau)]P = \sum_{j=0}^{L-1} [n_j] \tau^j (P).$$

From Equation (4.1) we see that doubling a point (or any scalar multiplication by powers of 2) can be performed using the 2-power Frobenius map on E, denoted by τ. Basically, to compute $[n]P$, instead of representing n in binary and computing $[n]P$ using a “double-and-(add or subtract)” algorithm, we represent n as a τ-adic expansion, and compute the scalar point multiplication using a "τ-and-(add or subtract)" algorithm, thereafter called only "τ-and-add" for simplicity. In other words, the "τ-and-add" algorithm, is based on the well known "double-and-add" algorithm, but the τ map (which has almost negligible cost if the field is represented in normal basis) replaces doublings. Hence, the total cost of $[n]P$ using the τ-and-add algorithm is w' group operations, where w' is the number of nonzero coefficients in the τ-adic representation of n.

Since the cost of $[n(\tau)]P$ roughly represents the cost of additions, the smaller the number of nonzero coefficients in $n(\tau)$ is, the more efficient $[n(\tau)]P$ will be performed. R. Avanzi, C. Heuberger and H. Prodinger [4] showed that the number of nonzero coefficients in τ-NAF expansions is *minimal* among all the representations with coefficients in $\{-1, 0, 1\}$.

J. Solinas [50, 51] gave an algorithm in the case of anomalous binary curves to transform an integer n into a τ-NAF (this algorithm was generalised in [31, 48]). The output of the algorithm proposed by Solinas is about twice as long as the ordinary NAF of n, but we can use the idea of Meier and Staffelbach [36] to reduce the τ-NAF expansion modulo $(\tau^m - 1)$ or modulo $(\tau^m - 1)/(\tau - 1)$, which is called reduced τ-NAF by Solinas (see also Section 4.2.7). In the reduced τ-NAF, the number of nonzero coefficients (i.e., w') is $L/3$, where L is the length of the (reduced) τ-NAF of n (see [4]).

If we use the NAF binary representation of n in the double-and-add algorithm, the number of nonzero bits (i.e., w) is approximately $m/3$ (see, for example, [54, pg 267]).

As a result, the total cost of $[n]P$ using the (binary) NAF representation of n in the double-and-add algorithm is $m + m/3 = 4m/3$ and when $L \approx m$, the total cost of $[n]P$ using the reduced τ-NAF representation of n in the τ-and-add algorithm is $m/3$ group operations (under the assumption that τ operation is cost-free), which means that in practice, the computation of $[n(\tau)]P$ can be 4 times faster than the standard double-and-add scalar multiplication.

W. Meier and O. Staffelbach [36] propose an algorithm for anomalous curves E defined over $\mathbb{F}_2$ and regarded as curves over the extension field $\mathbb{F}_{2^m}$ which computes multiples of arbitrary points on E and takes $m/2$ group operations. If we use the reduced τ-NAF representation of n in the τ-and-add algorithm we get the result of $[n]P$ 50% faster.

It is also possible to use point halving to perform fast scalar multiplications on generic elliptic curves over binary fields. However, although point halving is faster than doubling, it is more expensive than applying τ map. R. Avanzi, M. Ciet, and F. Sica [1] combined Frobenius operations with one point halving to compute scalar multiplications on Koblitz curves using on average 14% less group additions than with the usual τ-and-add method without increasing memory usage. Later, Avanzi et al. [4] improved that result, obtaining a scalar multiplication which requires on average 25% less group operations than the Frobenius method (i.e., using only τ-and-add).

See also [3, 5, 6] for recent developments in the arithmetic of Koblitz curves, specially regarding sublinear complexity scalar multiplication methods with essentially no precomputation.

4.2.4 Writing a general τ-adic as $x + y\tau$ for $x, y \in \mathbb{Z}$

Since τ satisfies a quadratic characteristic polynomial it follows that every Frobenius expansion of length L is equivalent to one of the form $x + y\tau$ for some uniquely determined $x, y \in \mathbb{Z}$. For our applications it is necessary to bound the sizes of x and y. In the following sections, we compute bounds for x and y using different techniques. The reader will notice that we start with a crude estimation and then we go towards sharper bounds.

4.2.4.1 $a = 3.7$ and $b = 2.6$

First we need the following Lemma, originally stated by Solinas [51]:

Lemma 4.2.2 Given the recurrence:

$$U_{i+1} = tU_i - 2U_{i-1} \text{ where } U_0 = 0, U_1 = 1 \ , \tag{4.5}$$

we have:

$$\tau^i \equiv U_i\tau - 2U_{i-1} \text{ for all } i > 0 \tag{4.6}$$

Proof. We prove by induction in i.

1. Base:

   $i = 1$. We have $\tau^1 = U_1\tau - 2U_0$.

2. Inductive step:

   We suppose that Equation (4.6) is true for $i = n - 1$ (that is: $\tau^{n-1} = U_{n-1}\tau - 2U_{n-2}$) and prove that the result holds for $i = n$ (that is:

$\tau^n = U_n\tau - 2U_{n-1})$.

$$\begin{aligned}
\tau^{n-1} &= U_{n-1}\tau - 2U_{n-2} && \text{induction hypothesis}\\
\tau^{n} &= U_{n-1}\tau^2 - 2U_{n-2}\tau && \text{multiplying induction hypothesis by } \tau\\
&= (t\tau - 2)U_{n-1} - 2U_{n-2}\tau && \text{since } \tau^2 \equiv t\tau - 2\\
&= (tU_{n-1} - 2U_{n-2})\tau - 2U_{n-1} && \text{grouping terms}\\
&= U_n\tau - 2U_{n-1} && \text{from Equation (4.5)}
\end{aligned}$$

□

Let $n \in \mathbb{N}$. Then there exist bounds $a, b \in \mathbb{N}$ such that for all $\alpha(\tau) = \sum_{i=0}^{n} \alpha_i\tau^i$, $\alpha_i \in \{-1, 0, 1\}$ it is always possible to write:

$$\alpha(\tau) \equiv x + y\tau \ ,$$

for some $x, y \in \mathbb{Z}$ such that $|x| < a$ and $|y| < b$. We now give values for a and b as functions of n.

Solving the recurrence relation (see [13]) in Equation (4.5), we get:

$$U_k = \tfrac{1}{\sqrt{-7}}\left(\tfrac{t+\sqrt{-7}}{2}\right)^k - \tfrac{1}{\sqrt{-7}}\left(\tfrac{t-\sqrt{-7}}{2}\right)^k,$$

for an integer $k > 0$.

Using the statement that the modulus of a complex number $x + y\mathbf{i}$ is $\sqrt{x^2 + y^2}$, we have:

$$\begin{aligned}
|U_k| &\leqslant \tfrac{1}{\sqrt{0^2+\sqrt{7}^2}}\sqrt{(\tfrac{t}{2})^2 + (\tfrac{\sqrt{7}}{2})^2}^k + \tfrac{1}{\sqrt{0^2+\sqrt{7}^2}}\sqrt{(\tfrac{t}{2})^2 + (\tfrac{\sqrt{7}}{2})^2}^k\\
&\leqslant \tfrac{1}{\sqrt{7}}\sqrt{2}^k + \tfrac{1}{\sqrt{7}}\sqrt{2}^k
\end{aligned}$$

Hence:

$$|U_k| \leqslant \frac{2\sqrt{2^k}}{\sqrt{7}} \tag{4.7}$$

We want to prove that: $|x| < 3.7(\sqrt{2})^n$ and $|y| < 2.6(\sqrt{2})^n$. We prove it by induction in n.

1. Base:

   $n = 0$. We have $\sum_{i=0}^{0} \alpha_i \tau^i = \alpha_0 \tau^0 = \alpha_0$.

   So, $\alpha_0 = |x| < a = 3.7(\sqrt{2})^0$, since $\alpha_i \in \{-1, 0, 1\}$.

   $n = 1$. We have $\sum_{i=0}^{1} \alpha_i \tau^i = \alpha_0 \tau^0 + \alpha_1 \tau^1 = \alpha_0 + \alpha_1 \tau$.

   So $\alpha_0 = x < 3.7(\sqrt{2})^0$ and $\alpha_1 = y < 2.6(\sqrt{2})^1$, since $\alpha_i \in \{-1, 0, 1\}$.

2. Inductive step:

   We take $\sum_{i=0}^{n-1} \alpha_i \tau^i = x' + y'\tau$. By hypothesis, it is true that:

$$|x'| < 3.7\sqrt{2^{n-1}} \tag{4.8}$$

   And:

$$|y'| < 2.6\sqrt{2^{n-1}} \tag{4.9}$$

   So, we need to prove that the result holds for n. First, we add another term to $\sum_{i=0}^{n-1} \alpha_i \tau^i$, as follows:

$$\begin{aligned} \textstyle\sum_{i=0}^{n} \alpha_i \tau^i &= x' + y'\tau + \alpha_n \tau^n \\ &= x' + y'\tau + \alpha_n \big(U_n \tau - 2U_{n-1}\big) \\ &= (x' - 2\alpha_n U_{n-1}) + (y' + \alpha_n U_n)\tau = x + y\tau \end{aligned}$$

First, for x (using (4.7) and (4.9)):

$$\begin{aligned}|x| &= |x' - 2\alpha_n U_{n-1}| \\ &\leqslant |x'| + 2|\alpha_n||U_{n-1}| \\ &\leqslant 3.7\sqrt{2^{n-1}} + 2\tfrac{2\sqrt{2^{n-1}}}{\sqrt{7}} = \left(\tfrac{3.7}{\sqrt{2}} + 2(\tfrac{2}{\sqrt{2}\sqrt{7}})\right)\sqrt{2^n} \\ &< 3.7\sqrt{2^n}\end{aligned}$$

Now, for y (using (4.7) and (4.8)):

$$\begin{aligned}|y| &= |y' + \alpha_n U_n| \\ &\leqslant |y'| + |\alpha_n U_n| \\ &\leqslant 2.6\sqrt{2^{n-1}} + 2\tfrac{\sqrt{2^n}}{\sqrt{7}} = \left(\tfrac{2.6}{\sqrt{2}} + \tfrac{2}{\sqrt{7}}\right)\sqrt{2^n} \\ &< 2.6\sqrt{2^n}\end{aligned}$$

The values of a, b can be slightly improved when we treat $z = \sum_{i=0}^{n} \alpha_i \tau^i$ as a complex number, following the idea proposed by James McKee.

4.2.4.2 $a = 3.65$ and $b = 2.59$

We know that τ satisfies the characteristic polynomial $\tau^2 - t\tau + 2 = 0$, for $t = \pm 1$. One can interpret τ as the complex number:

$$\tau = \frac{t \pm \sqrt{-7}}{2}.$$

Hence, the norm of τ is:

$$|\tau| = \left|\frac{t \pm \sqrt{7}\mathbf{i}}{2}\right| = \sqrt{\frac{t^2 + \sqrt{7}^2}{4}} = \sqrt{\frac{1+7}{4}} = \sqrt{2}. \qquad (4.10)$$

Let $z = \sum_{i=0}^{n} \alpha_i \tau^i$ be a complex number. The norm of z is:

$$|z| = \left|\sum_{i=0}^{n} \alpha_i \tau^i\right| \leqslant \sum_{i=0}^{n} |\alpha_i||\tau| \leqslant \sum_{i=0}^{n} \sqrt{2}^i$$

Using the formula for the sum of $n+1$ first terms of a geometric progression, we have:

$$|z| \leqslant \sum_{i=0}^{n} \sqrt{2}^i = \frac{\sqrt{2}^{n+1} - 1}{\sqrt{2} - 1}$$

and

$$|z|^2 < \left(\frac{\sqrt{2}^{n+1}(\sqrt{2}+1)}{(\sqrt{2}-1)(\sqrt{2}+1)}\right)^2 = 2^{n+1}(\sqrt{2}+1)^2. \tag{4.11}$$

Now, we use the fact that z can be written as $x + y\tau$:

$$z = x + y\left(\frac{t \pm \sqrt{-7}}{2}\right) = \left(x \pm \frac{y}{2}\right) \pm \frac{\sqrt{-7}}{2} y.$$

And hence:

$$|z| = \sqrt{x^2 \pm xy + 2y^2}.$$

Which leads to:

$$|z|^2 = x^2 \pm xy + 2y^2. \tag{4.12}$$

From Equations (4.11) and (4.12) we have:

$$x^2 \pm xy + 2y^2 < 2^{n+1}(\sqrt{2}+1)^2.$$

Completing the squares, we have:

$$x^2 \pm xy + 2y^2 = 2\left(y \pm \frac{x}{4}\right)^2 + \frac{7x^2}{8} < 2^{n+1}(\sqrt{2}+1)^2.$$

Since the first term on the left hand side of the inequality (i.e., $2\left(y \pm \frac{x}{4}\right)^2$) is a square and hence $\geqslant 0$, we can write:

$$\frac{7x^2}{8} < 2^{n+1}(\sqrt{2}+1)^2.$$

Therefore:

$$\begin{aligned} |x| &< \sqrt{\tfrac{8}{7}2^{n+1}(\sqrt{2}+1)^2} \\ &< 3.65\sqrt{2^n}. \end{aligned}$$

Now, completing the squares in a different way, we have:

$$x^2 \pm xy + 2y^2 = \left(x \pm \frac{y}{2}\right)^2 + \frac{7y^2}{4} < 2^{n+1}(\sqrt{2}+1)^2.$$

Again, since the first term on the left hand side of the inequality (i.e., $\left(x \pm \frac{y}{2}\right)^2$) is a square and hence $\geqslant 0$, we can write:

$$\frac{7y^2}{4} < 2^{n+1}(\sqrt{2}+1)^2.$$

Therefore:

$$\begin{aligned} |y| &< \sqrt{\tfrac{4}{7}2^{n+1}(\sqrt{2}+1)^2} \\ &< 2.59\sqrt{2^n}. \end{aligned}$$

Note that the values for a and b are slightly smaller than the ones we had found before.

However, it is possible to get even smaller values for a and b by pairing consecutive terms in the τ-adic expansion together and using a worst-case bound for the sum of two terms. Similarly, one can group the terms together in threes, fours, fives etc. to get even smaller values for a, b. We also thank James McKee for that idea.

4.2.4.3 $a = 3.03$ and $b = 2.14$

We begin by writing $z = \alpha_0 + \alpha_1\tau + \alpha_2\tau^2 + \alpha_3\tau^3 + \ldots + \alpha_n\tau^n$.

a) $n \equiv 1 \pmod 2$

   Pairing consecutive terms, leads to

   $$z = (\alpha_0+\alpha_1\tau)+(\alpha_2+\alpha_3\tau)\tau^2+(\alpha_4+\alpha_5\tau)\tau^4+\ldots+(\alpha_{n-1}+\alpha_n\tau)\tau^{n-1}.$$

   There is a bound $B \in \mathbb{N}$ such that $|\alpha_i + \alpha_{i+1}\tau| \leqslant B$ for all i.

   Hence, we can rewrite as:

   $$|z| \leqslant B(1 + |\tau^2| + |\tau^4| + |\tau^6| + \ldots + |\tau^{n-1}|).$$

   From Equation(4.10), the norm of τ is $\sqrt{2}$, so:

   $$|z| \leqslant B(1+2+2^2+\ldots+2^{\frac{n-1}{2}}) = B\left(\frac{2^{\frac{n-1}{2}+1}-1}{2-1}\right) = B(2^{\frac{n+1}{2}}-1) < B(2^{\frac{n+1}{2}}).$$

   Therefore

   $$|z|^2 < B^2(2^{n+1}). \tag{4.13}$$

   Now we compute the worst-case bound for B, which occurs when $\alpha_i = \alpha_{i+1} = 1$ and $\tau = \frac{1+\sqrt{-7}}{2}$:

   $$B = |(\alpha_i+\alpha_{i+1}\tau)| \leqslant \left|1 + \frac{1+\sqrt{-7}}{2}\right| = \left|\frac{3+\sqrt{-7}}{2}\right| = \sqrt{\frac{3^2+\sqrt{7}^2}{4}} = 2.$$

   Using the bound B in Equation (4.13):

   $$|z|^2 < 4(2^{n+1}). \tag{4.14}$$

From Equations (4.12) and (4.14):

$$x^2 \pm xy + 2y^2 < 4(2^{n+1}).$$

Now, completing the squares in the same way we did in the last section, we end up with:

$$\begin{aligned} |x| &< \sqrt{4(\tfrac{8}{7}2^{n+1})} \\ &< 3.03\sqrt{2^n}. \end{aligned}$$

and

$$\begin{aligned} |y| &< \sqrt{4(\tfrac{4}{7}2^{n+1})} \\ &< 2.14\sqrt{2^n}. \end{aligned}$$

b) $n \equiv 0 \pmod 2$

We rewrite:

$$z = a_0 + z'\tau.$$

Where

$$z' = (\alpha_1 + \alpha_2\tau) + (\alpha_3 + \alpha_4\tau)\tau^2 + ... + (\alpha_{n-1} + \alpha_n\tau)\tau^{n-2}.$$

So

$$z'\tau = z - \alpha_0.$$

We call

$$z'' = (z - \alpha_0). \tag{4.15}$$

It follows that

$$|z'||\tau| = |z - \alpha_0| = |z''|.$$

And since $|\tau| = \sqrt{2}$, it follows that

$$|z''| \leqslant \sqrt{2}|z'| \tag{4.16}$$

Since z' has degree $n-1$, by the previous case it follows that:

$$|z'| < 2\sqrt{2^n}.$$

Then using the above result with Equation (4.16):

$$|z''| \quad < \sqrt{2}(2\sqrt{2^n}) = 2(2^{\frac{n+1}{2}}).$$

Now, using the fact that $z'' = x'' + y''\tau$, it follows from the previous result that:

$$\begin{aligned} |x''| &\quad < 3.03\sqrt{2^n} \\ |y''| &\quad < 2.14\sqrt{2^n}. \end{aligned}$$

Finally, since $z = x + y\tau$, from Equation (4.15) we have

$$x'' + y''\tau = x + y\tau - \alpha_0 = (x - \alpha_0) + y\tau.$$

Since $\alpha_i = \{-1, 0, 1\}$ for any i, it follows that $|x|$ and $|x''|$ differ by at most 1. Therefore:

$$\begin{aligned} |x| &\quad < 1 + 3.03\sqrt{2^n} = \left(\frac{1}{\sqrt{2^n}} + 3.03\right)\sqrt{2^n} \\ |y| &\quad < 2.14\sqrt{2^n}. \end{aligned}$$

Now we group the terms in threes, to get even smaller bounds.

4.2.4.4 $a = 2.81$ and $b = 2$

We begin by writing $z = \alpha_0 + \alpha_1\tau + \alpha_2\tau^2 + \alpha_3\tau^3 + \ldots + \alpha_n\tau^n$.

a) $n \equiv 2 \pmod 3$

Pairing three consecutive terms, leads to

$$(\alpha_0+\alpha_1\tau+\alpha_2\tau^2)+(\alpha_3+\alpha_4\tau+\alpha_5\tau^2)\tau^3+\ldots+(\alpha_{n-2}+\alpha_{n-1}\tau+\alpha_n\tau^2)\tau^{n-2}.$$

There is a bound $C \in \mathbb{N}$ such that $|\alpha_i + \alpha_{i+1}\tau + \alpha_{i+2}\tau^2| \leqslant C$ for all i.

Hence, we can rewrite as:

$$|z| \leqslant C(1 + |\tau^3| + |\tau^6| + |\tau^9| + \ldots + |\tau^{n-2}|).$$

From Equation(4.10), the norm of τ is $\sqrt{2}$, so:

$$\begin{aligned} |z| &\leqslant C(1 + \sqrt{2}^3 + \sqrt{2}^6 + \sqrt{2}^9 + \ldots + \sqrt{2}^{n-2}) \\ &= C(1 + 2^{3/2} + (2^{3/2})^2 + (2^{3/2})^3 + \ldots + (2^{3/2})^{\frac{n-2}{3}}) \\ &= C\left(\tfrac{(2^{3/2})^{\frac{n-2}{3}+1}-1}{2\sqrt{2}-1}\right) = C\left(\tfrac{2^{\frac{n+1}{2}}-1}{2\sqrt{2}-1}\right) \\ &< \tfrac{C}{2\sqrt{2}-1}2^{\frac{n+1}{2}} \end{aligned}$$

Therefore

$$|z|^2 < \frac{C^2}{(2\sqrt{2}-1)^2}2^{n+1}. \tag{4.17}$$

Now we compute the worst-case bound for C, which occurs when $\alpha_i = -1$, $\alpha_{i+1} = \alpha_{i+2} = 1$ and $\tau = \frac{1+\sqrt{-7}}{2}$:

$$C \;= |(\alpha_i + \alpha_{i+1}\tau + \alpha_{i+2}\tau^2)| \leqslant \left|-1 + \tfrac{1+\sqrt{-7}}{2} + (\tfrac{1+\sqrt{-7}}{2})^2\right| = |-2 + \sqrt{-7}|$$

Hence

$$C \leqslant \sqrt{(-2)^2 + \sqrt{7}^2} = \sqrt{11}.$$

Using the bound C in Equation (4.17):

$$|z|^2 < \frac{11}{(2\sqrt{2}-1)^2}(2^{n+1}). \tag{4.18}$$

From Equations (4.12) and (4.18):

$$x^2 \pm xy + 2y^2 < \frac{11}{(2\sqrt{2}-1)^2}(2^{n+1}).$$

Now, completing the squares in the same way we did in the last section, we end up with:

$$\begin{aligned} |x| &< \sqrt{\tfrac{8}{7}\tfrac{11}{(2\sqrt{2}-1)^2}(2^{n+1})} \\ &< 2.75\sqrt{2^n}. \end{aligned}$$

and

$$\begin{aligned} |y| &< \sqrt{\tfrac{4}{7}\tfrac{11}{(2\sqrt{2}-1)^2}(2^{n+1})} \\ &< 1.94\sqrt{2^n}. \end{aligned}$$

b) $n \equiv 1 \pmod 3$

We can rewrite:

$$z = \alpha_0 + \alpha_1\tau + z'\tau^2.$$

Where

$$z' = (\alpha_2+\alpha_3\tau+\alpha_4\tau^2)+(\alpha_5+\alpha_6\tau+\alpha_7\tau^2)\tau^3+\ldots+(\alpha_{n-2}+\alpha_{n-1}\tau+\alpha_n\tau^2)\tau^{n-4}.$$

So

$$z'\tau^2 = z - \alpha_0 - \alpha_1\tau.$$

Then, we call

$$z'' = (z - \alpha_0 - \alpha_1 \tau). \tag{4.19}$$

It follows that

$$|z'||\tau|^2 = |z - \alpha_0 - \alpha_1 \tau| = |z''|.$$

And using the fact that $|\tau| = \sqrt{2}$:

$$|z''| \leqslant 2|z'| \tag{4.20}$$

Since z' has degree $n-2$, by the previous case it follows that:

$$|z'| < \left(\frac{\sqrt{11}}{2\sqrt{2}-1} \right) 2^{\frac{n-1}{2}}.$$

Then using the above result with Equation (4.20):

$$|z''| \quad < 2 \left(\tfrac{\sqrt{11}}{2\sqrt{2}-1} \right) 2^{\frac{n-1}{2}} = \left(\tfrac{\sqrt{11}}{2\sqrt{2}-1} \right) 2^{\frac{n+1}{2}}$$

Now, using the fact that $z'' = x'' + y''\tau$, it follows from the previous result that:

$$\begin{aligned} |x''| &\quad < 2.75\sqrt{2^n} \\ |y''| &\quad < 1.94\sqrt{2^n}. \end{aligned}$$

Finally, since $z = x + y\tau$, from Equation (4.19) we have

$$x'' + y''\tau = x + y\tau - \alpha_0 - \alpha_1\tau = (x - \alpha_0) + (y - \alpha_1)\tau.$$

Since $\alpha_i = \{-1, 0, 1\}$ for any i, it follows that $|x|$ and $|x''|$ differ by at most 1. The same occurs between $|y|$ and $|y''|$. Therefore:

$$\begin{aligned} |x| &< 1 + 2.75\sqrt{2^n} = \left(\frac{1}{\sqrt{2^n}} + 2.75\right)\sqrt{2^n} \\ |y| &< 1 + 1.94\sqrt{2^n} = \left(\frac{1}{\sqrt{2^n}} + 1.94\right)\sqrt{2^n}. \end{aligned}$$

c) $n \equiv 0 \pmod 3$

We can rewrite:

$$z = \alpha_0 + z'\tau.$$

Where

$$z' = (\alpha_1+\alpha_2\tau+\alpha_3\tau^2)+(\alpha_4+\alpha_5\tau+\alpha_6\tau^2)\tau^3+\ldots+(\alpha_{n-2}+\alpha_{n-1}\tau+\alpha_n\tau^2)\tau^{n-3}.$$

So

$$z'\tau = z - \alpha_0.$$

We call

$$z'' = (z - \alpha_0). \tag{4.21}$$

It follows that

$$|z'||\tau| = |z - \alpha_0| = |z''|.$$

Taking $|\tau| = \sqrt{2}$:

$$|z''| \leqslant \sqrt{2}|z'| \tag{4.22}$$

Since z' has degree $n-1$, by the previous case it follows that:

$$|z'| < \sqrt{11}\left(\frac{(2\sqrt{2})^{\frac{n}{3}}}{2\sqrt{2}-1}\right) = \left(\frac{\sqrt{11}}{2\sqrt{2}-1}\right)2^{\frac{n}{2}}.$$

Now, using the fact that $z'' = x'' + y''\tau$, it follows from the previous result that:

$$\begin{aligned} |x''| &< 2.75\sqrt{2^n} \\ |y''| &< 1.94\sqrt{2^n}. \end{aligned}$$

Finally, since $z = x + y\tau$, from Equation (4.21) we have

$$x'' + y''\tau = x + y\tau - \alpha_0 = (x - \alpha_0) + y\tau.$$

Since $\alpha_i = \{-1, 0, 1\}$ for any i, it follows that $|x|$ and $|x''|$ differ by at most 1. Therefore:

$$\begin{aligned} |x| &< 1 + 2.75\sqrt{2^n} \\ |y| &< 1.94\sqrt{2^n}. \end{aligned}$$

Finally, we can state the following theorem:

Theorem 4.2.2. *Let $n > 8$ and let $\alpha(\tau) = \sum_{i=0}^{n} \alpha_i \tau^i$, for $\alpha_i \in \{-1, 0, 1\}$. It is always possible to write:*

$$\alpha(\tau) \equiv x + y\tau\ ,$$

for some $x, y \in \mathbb{Z}$ such that $|x| < a = 2.81\sqrt{2^n}$ and $|y| < b = 2\sqrt{2^n}$.

Proof. From the results of items a), b) and c) above, the worst-case bounds for x and y are:

$$\begin{aligned} |x| &< 1 + 2.75\sqrt{2^n} = \left(\frac{1}{\sqrt{2^n}} + 2.75\right)\sqrt{2^n} \\ |y| &< 1 + 1.94\sqrt{2^n} = \left(\frac{1}{\sqrt{2^n}} + 1.94\right)\sqrt{2^n}. \end{aligned}$$

Since $\frac{1}{\sqrt{2^n}} < 0.06$ when $n > 8$, the proof is complete. $\square$

We remark that one could try to prove even smaller values for a and b, following the idea above and grouping the terms together in fours, fives, etc.

In Section 4.2.6 we conjecture smaller values for a and b (namely $a = 2.28\sqrt{2^n}$ and $b = 1.8\sqrt{2^n}$). Our conjecture is motivated by experimental results.

4.2.5 Writing a τ-NAF as $x + y\tau$ for $x, y \in \mathbb{Z}$

We can use the same idea to give more precise values for the bounds a, b when dealing with τ-NAFs.

We recall that $z = \alpha_0 + \alpha_1\tau + \alpha_2\tau^2 + \alpha_3\tau^3 + \ldots + \alpha_n\tau^n$.

a) $n \equiv 1 \pmod 2$

Pairing consecutive terms, leads to

$$(\alpha_0 + \alpha_1\tau) + \tau^2(\alpha_2 + \alpha_3\tau) + \tau^4(\alpha_4 + \alpha_5\tau) + \ldots + \tau^{n-1}(\alpha_{n-1} + \alpha_n\tau).$$

Note that, since z is a τ-NAF, either α_i or α_{i+1} is zero. There is a bound $D \in \mathbb{N}$ such that $|\alpha_i + \alpha_{i+1}\tau| \leqslant D$ for all i. As a result, the bound D will be smaller than B, which will lead to smaller bounds.

We can rewrite as:

$$|z| \leqslant D(1 + |\tau^2| + |\tau^4| + |\tau^6| + \ldots + |\tau^{n-1}|)$$

Now we compute the worst-case bound for D, which occurs when $\alpha_i =$

0, $\alpha_{i+1} = 1$ and $\tau = \frac{1+\sqrt{-7}}{2}$:

$$D = |(\alpha_i+\alpha_{i+1}\tau)| \leqslant \left|0 + \frac{1+\sqrt{-7}}{2}\right| = \left|\frac{1+\sqrt{-7}}{2}\right| = \sqrt{\frac{1^2+\sqrt{7}^2}{4}} = \sqrt{2}.$$

It follows that:

$$|z|^2 < 2(2^{n+1}). \tag{4.23}$$

Using Equations (4.12) and (4.23):

$$x^2 - xy + 2y^2 < 2(2^{n+1}).$$

Now, completing the squares in the same way we did in the last section, we end up with:

$$\begin{aligned} |x| &< \sqrt{2(\tfrac{8}{7}2^{n+1})} \\ &< 2.14\sqrt{2^n}. \end{aligned}$$

and

$$\begin{aligned} |y| &< \sqrt{2(\tfrac{4}{7}2^{n+1})} \\ &< 1.52\sqrt{2^n}. \end{aligned}$$

b) $n \equiv 0 \pmod 2$

We rewrite:

$$z = a_0 + z'\tau.$$

Where

$$z' = (\alpha_1 + \alpha_2\tau) + (\alpha_3 + \alpha_4\tau)\tau^2 + ... + (\alpha_{n-1} + \alpha_n\tau)\tau^{n-2}.$$

So

$$z'\tau = z - \alpha_0.$$

Then, we call

$$z'' = (z - \alpha_0). \tag{4.24}$$

It follows that

$$|z'||\tau| = |z - \alpha_0| = |z''|.$$

Recall that $|\tau| = \sqrt{2}$. Hence:

$$|z''| \leqslant \sqrt{2}|z'| \tag{4.25}$$

Since z' has degree $n-1$, by the previous case it follows that:

$$|z'| < (\sqrt{2})2^{\frac{n}{2}}.$$

Then using the above result with Equation (4.25):

$$|z''| \quad < (\sqrt{2})(\sqrt{2})2^{\frac{n}{2}} = \sqrt{2}(2^{\frac{n+1}{2}})$$

Now, using the fact that $z'' = x'' + y''\tau$, it follows from the previous result that:

$$\begin{aligned} |x''| &< 2.14\sqrt{2^n} \\ |y''| &< 1.52\sqrt{2^n}. \end{aligned}$$

Finally, since $z = x + y\tau$, from Equation (4.24) we have

$$x'' + y''\tau = x + y\tau - \alpha_0 = (x - \alpha_0) + y\tau.$$

Since $\alpha_i = \{-1, 0, 1\}$ for any i, it follows that $|x|$ and $|x''|$ differ by at most 1. Therefore:

$$\begin{aligned} |x| &< 1 + 2.14\sqrt{2^n} = \left(\tfrac{1}{\sqrt{2^n}} + 2.14\right)\sqrt{2^n} \\ |y| &< 1.52\sqrt{2^n}. \end{aligned}$$

Finally, we can state the following theorem:

Theorem 4.2.3. *Let $n > 8$ and let $\alpha(\tau) = \sum_{i=0}^{n} \alpha_i \tau^i$, for $\alpha_i \in \{-1, 0, 1\}$ be a τ-NAF expansion. It is always possible to write:*

$$\alpha(\tau) \equiv x + y\tau \ ,$$

for some $x, y \in \mathbb{Z}$ such that $|x| < a = 2.20\sqrt{2^n}$ and $|y| < b = 1.52\sqrt{2^n}$.

Proof. From the results of items a), b) above, the worst-case bounds for x and y are:

$$\begin{aligned} |x| &< 1 + 2.14\sqrt{2^n} = \left(\tfrac{1}{\sqrt{2^n}} + 2.14\right)\sqrt{2^n} \\ |y| &< 1.52\sqrt{2^n}. \end{aligned}$$

Since $\frac{1}{\sqrt{2^n}} < 0.06$ when $n > 8$, the proof is complete. □

Now we conjecture smaller values for a, b based on the distribution of random τ-adics and random τ-NAFs.

4.2.6 Using the Distribution of Random τ-adics to obtain Smaller Bounds

We recall that in Theorems 4.2.2 and 4.2.3 we proved bounds for x and y. Now, based on experimental results, we conjecture that such bounds can be improved to smaller values. We experimented in MAGMA [10] generating 10,000 random τ-adics (and random τ-NAFs) of length L and then we converted each one to $x + y\tau$ using the algorithm presented in [12]. By analysing the pictures of each distribution, we could state a more restrictive conjecture for the bounds of a and b.

4.2.6.1 Distribution of Random τ-adics as $x + y\tau$ expressions

Figures 4.2, 4.3, 4.4 and 4.5 show the distribution of 10,000 random τ-adics, for L respectively equal to 30, 40, 50 and 160. In all figures, the values in x go from $-a$ to a, using $a = 3\sqrt{2^L}$ and the values in y go from $-b$ to b, using $b = 2\sqrt{2^L}$. Note that such values for a and b are an approximation to integers of the values proved in Theorem 4.2.2.

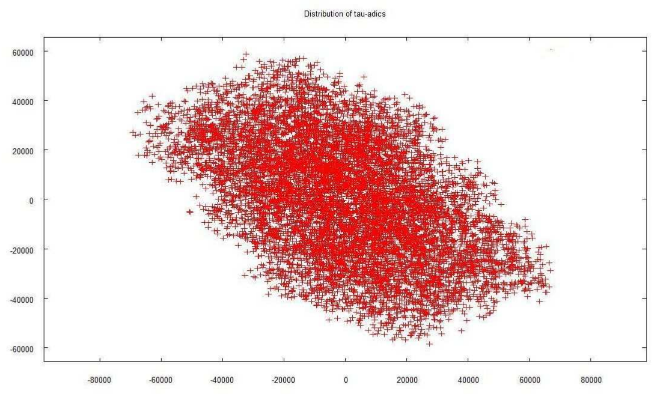

Figure 4.2: Distribution of 10,000 random τ-adics for $L \leqslant 30$. The bounds for axis x and y are respectively $(-a, a)$ and $(-b, b)$, for $a = 3\sqrt{2^L}$ and $b = 2\sqrt{2^L}$.

Notice that the distribution is centered in a region of smaller area than $4ab$ and it seems to be close to uniform away from the edges. Furthermore, the symmetry under $(x, y) \to (-x, -y)$ (i.e., 180° rotation) is obvious. For $L = 30$, we can check that for both x and y positive or both negative, there is no point after certain size of x and y (note the the picture looks like a "sausage"). This can be explained when we convert points in this

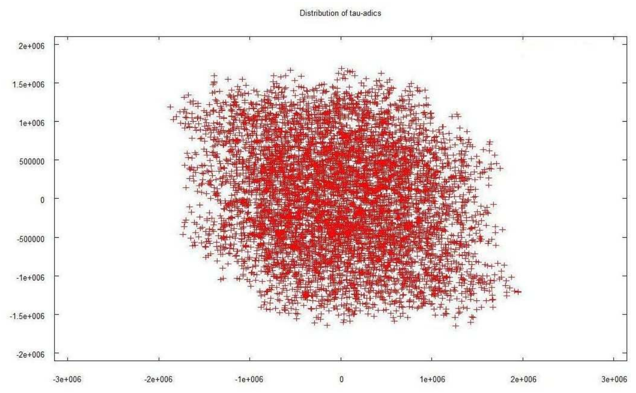

Figure 4.3: Distribution of 10,000 random τ-adics for $L \leqslant 40$. The bounds for axis x and y are respectively $(-a, a)$ and $(-b, b)$, for $a = 3\sqrt{2^L}$ and $b = 2\sqrt{2^L}$.

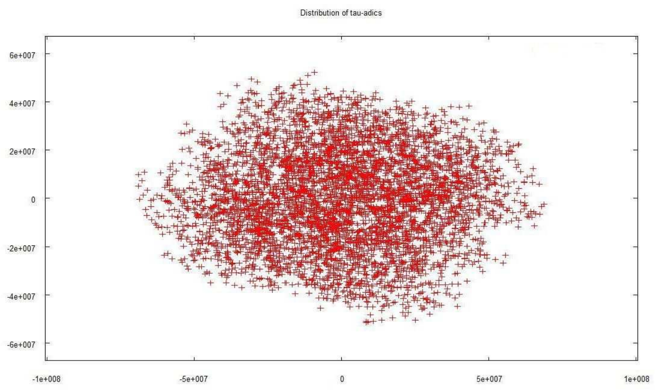

Figure 4.4: Distribution of 10,000 random τ-adics for $L \leqslant 50$. The bounds for axis x and y are respectively $(-a, a)$ and $(-b, b)$, for $a = 3\sqrt{2^L}$ and $b = 2\sqrt{2^L}$.

empty region back to a τ-adic with coefficients only in $\{-1, 0, 1\}$ (using Algorithm 4). For example, take $x = 50000$ and $y = 30000$. Check in Figure 4.2 that point (x, y) is empty. When we convert $50000 + 30000\tau$ to a τ-adic with coefficients in $\{-1, 0, 1\}$ using Algorithm 4, we get $\tau^{31} - \tau^{30} - \tau^{29} - \tau^{24} + \tau^{22} + \tau^{21} - \tau^{15} - \tau^{13} + \tau^{12} + \tau^{10} + \tau^7 - \tau^4$ which explains why the region is empty, since we are using $L = 30$. The same occurs with $x = -50000$ and $y = -30000$. For $L = 40$ and $L = 50$ this behaviour does not occur and the distribution is centered. Note that for $L = 160$ the same "sausage-behaviour" occurs.

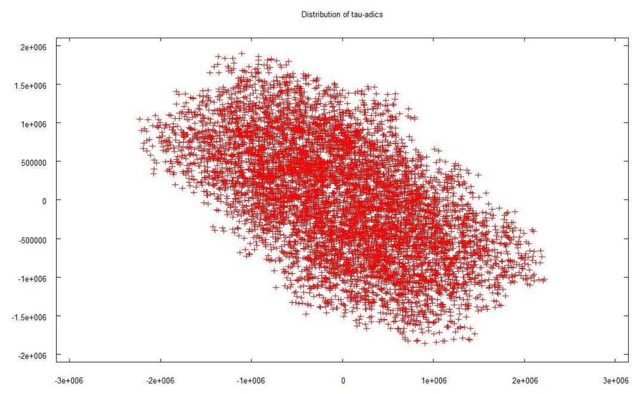

Figure 4.5: Distribution of 10,000 random τ-adics for $L \leqslant 160$. Each value is divided by 2^{60}. The bounds for axis x and y are respectively $(-A, A)$ and $(-B, B)$, for $A = 3\sqrt{2^L}$ and $B = 2\sqrt{2^L}$. Note that the axes have been rescaled by a factor 2^{60}.

A possible explanation for the 'sausage-shape' comes when we plot the distribution for $L = 29$. The picture looks more centered (similar to $L = 40$). Going from the picture for $L = 29$ to the picture for $L = 30$ is equivalent to shift the original picture to the positions τ^{30} and $-\tau^{30}$. It happens that when we convert τ^{30} and $-\tau^{30}$ to coefficients in $\mathbb{Z}$ we get $17282 - 24475\tau$ and $-17282 + 24475\tau$ respectively. Note that the new points (τ^{30} and $-\tau^{30}$) are in the right-bottom and left-top corners of the original picture. Hence, when we shift the original picture to the new points, we got Figure 4.2, with its 'sausage-shape'. Doing the same for $L = 39$, when we plot τ^{40} and $-\tau^{40}$ we get $49138 - 753027\tau$ and $-49138 + 753027\tau$. Since those points are close to the y axis, we do not get a 'sausage-shape' for $L = 40$.

An alternative explanation for the 'sausage' forms, according to Roberto Avanzi: On the Gauss plane, going from τ-adics of length L to τ-adics of length $L + 1$ is a multiplication of the previous graph by τ and addition some detail corresponding to the least significant digit. This can be viewed as added detail to the pictures. So we essentially get a rotating figure, once the size has been normalised. Now, going from the Gauss plane to $(1, \tau)$-

coordinates is just a linear transformation of the plane.

Now we state the following conjecture, based on the pictures:

Conjecture 1. *Let $n(\tau) = \sum_{i=0}^{L} \alpha_i \tau^i$, for $\alpha \in \{-1, 0, 1\}$. Let x, y be integers. For practical values of L it is always possible to write:*

$$n(\tau) \equiv x + y\tau \ ,$$

with $|x| < a = 2.28\sqrt{2^L}$ and $|y| < b = 1.8\sqrt{2^L}$.

Evidence for the conjecture - From the distributions of random $x+y\tau$ (see Figures 4.2, 4.3 and 4.4) we observed that the maximum point is $< 0.76 * a$ in the x-direction and $< 0.9 * b$ in the y-direction. As a result, we can define new bounds a and b, namely $a = 0.76 * (3\sqrt{2^L})$ and $b = 0.9 * (2\sqrt{2^L})$.

We observe that a better choice would not be a box but some curve that fits better the region of distribution. The most obvious choice would be an ellipse. Alternatively, one can change coordinates to give a box that is aligned with the major axis of the ellipse. We leave this as an open problem.

4.2.6.2 Distribution of Random τ-NAFs as $x + y\tau$ expressions

We repeated the same experiment, but now generating random τ-NAFs. We could observe that we can take even smaller bounds when working with τ-NAFs. Figure 4.6 shows the distribution of 10,000 random τ-NAFs.

So we state the following conjecture:

Conjecture 2. *Let $n(\tau) = \sum_{i=0}^{L} \alpha_i \tau^i$, for $\alpha \in \{-1, 0, 1\}$ be a τ-NAF expansion. Let x, y be integers. For practical values of L it is always possible*

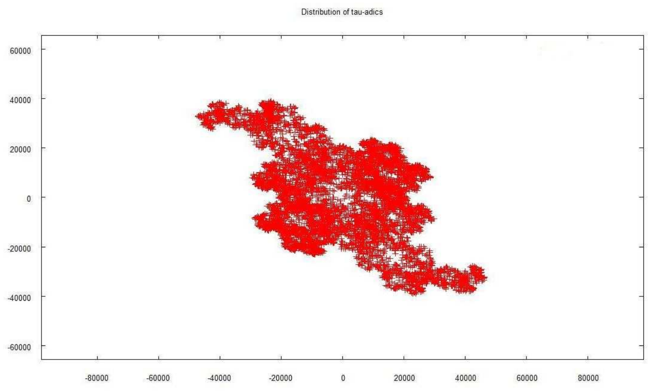

Figure 4.6: Distribution of 10,000 random τ-NAFs for $L \leqslant 30$. The bounds for axis x and y are respectively $(-a, a)$ and $(-b, b)$, for $a = 3\sqrt{2^L}$ and $b = 2\sqrt{2^L}$.

to write:

$$n(\tau) \equiv x + y\tau \ ,$$

with $|x| < a = (0.51) \times 3\sqrt{2^L} = 1.53\sqrt{2^L}$ *and* $|y| < b = (0.62) \times 2\sqrt{2^L} = 1.24\sqrt{2^L}$.

Again, we observe that the best choice would be an ellipse instead of a box or even a box aligned with the major axis of the ellipse.

4.2.6.3 About Distributions

We remark that one cannot use the picture of τ-adic distributions to estimate the number of equivalence classes by just counting the number of points in the box in which the distribution lies. The quantity $4ab$ only gives an *upper bound* and it is clear from the pictures that the actual number of equivalence classes can be a lot smaller than that. As we observed before, the best choice would not be a box, but some curve which fits better in the distribution region. Note that not every point (x, y) (even in the most dense central part of the picture) necessarily arises from a τ-adic expansion. In other words, the picture is not solid, it has holes in it. This is clearly seen in the $L = 30$ and $L = 160$ distributions, which look like a "sausage", but it also occurs in the $L = 40$ and $L = 50$ distributions. Obviously, the holes can be explained due to the simulation, because we generated $10,000$ random τ-adics and the number of equivalence classes is much bigger (i.e., $\tilde{O}(2^L)$). Note that we can see some holes close to the centre of the region. This is because if we take $x + y\tau$ for very small x and y, the resultant τ-adic with coefficients in $\{-1, 0, 1\}$ has small degree and therefore, it is unlikely to occur unless we generate a sufficiently large number of random τ-adics.

However, there are also theoretical holes as we get closer to the edges of the region. For example, take the point $P_1 = 48950 - 7193\tau$ in Figure 4.2. It can be seen that P_1 lies inside the bounds of the region of points. If we convert P_1 to coefficients in $\{-1, 0, 1\}$ using Algorithm 4, we obtain τ^{31}, so P_1 does not arise when we take $L = 30$.

Furthermore, we remember from Table 4.1 in Section 4.2.2 that the number of P-equivalence classes for $L = 15$ is 160359. Using the bounds for τ-adic distribution, we compute $a = (0.76)3\sqrt{2^{15}} \approx 413$ and $b = (0.9)2\sqrt{2^{15}} \approx 326$ (we also repeated the experiment by choosing $10,000$ random τ-adics of length $L = 15$ and drew a picture, and we could check that the bounds hold). Hence, we have $4ab \approx 538552$, which is more than 3 times greater than the correct number of P-equivalence classes. Note that the pictures show the number of equivalence classes whereas Table 4.1 shows the number of P-equivalence classes, which naturally is smaller than the number of equivalence classes (see Definitions 19 and 20), but we believe that the main reason for the difference is the presence of holes, i.e., points that do not arise from a τ-adic of certain length. We leave as a open problem a better understanding of the distribution of τ-adics.

4.2.7 Converting Integers to Frobenius Expansions and Vice-Versa

Since point multiplication can be efficiently performed when one uses a Frobenius expansion instead of an integer, a common practice to improve efficiency is to use a Koblitz curve and convert integers into τ-adic expansions. We now briefly mention the algorithms to convert integers into Frobenius expansions.

J. Solinas [50, Algorithm 4, pg. 364] proposed an algorithm that converts $n = x + y\tau$ for some integers x, y into a Frobenius expansion in non-adjacent form (τ-NAF) equivalent to n. It is easy to see that when $y = 0$, the algorithm converts an integer ($n = x$) into a τ-NAF. That algorithm only involves elementary integer operations. One problem is that the resulting Frobenius expansion typically has degree much larger than m, so in the same paper there is an algorithm ([50, Algorithm 5, pg. 365]) which reduces the τ-NAF of n modulo $\tau^m - 1$ or modulo $(\tau^m - 1)/(\tau - 1)$, following an idea of Meier and Staffelbach [36] (Solinas call that output "reduced τ-NAF"). This latter algorithm requires many integer divisions and hence, can be costly.

According to N. Koblitz [33], H. Lenstra suggested that, instead of choosing a random integer and afterwards, convert it to a τ-adic for efficient purposes, one can start a protocol choosing a random τ-adic expansion. If a random τ-NAF is chosen, since it is unique, one can use it without caring which integer it represents.

For some applications, it may be necessary to convert a Frobenius expansion $\sum_i n_i \tau^i$ to an integer n (for example, Elliptic Curve Digital Signature Algorithm (ECDSA) requires both the integer and the scalar multiple). For such applications, it is necessary to revise the concept of *eigenvalue of Frobenius.*

Definition 21. *The Frobenius map in a cyclic group of order* $\mathbf{r}$ *has some eigenvalue* λ*, so that* $\tau(P) = [\lambda]P$*. This eigenvalue is a solution to the characteristic polynomial modulo* $\mathbf{r}$*.*

Once we compute the eigenvalue λ on $\langle P \rangle$, it follows that $n = \sum_i n_i \lambda^i \pmod{\mathbf{r}}$, where $\mathbf{r}$ is the order of P. In other words, this algorithm requires arithmetic modulo a large prime.

B. Brumley and K. Järvinen[12] propose a more efficient algorithm which takes a τ-adic of length L as input and computes the equivalent element $x + y\tau$, for some $x, y \in \mathbb{Z}$. Once $x + y\tau$ is computed, the equivalent integer n modulo $\mathbf{r}$ is easily obtained using $n = x + y\lambda \pmod{\mathbf{r}}$. A hardware implementation is provided in the paper and the authors claim that "the results suggest significant computational efficiency gains over previously documented methods".

4.2.8 Bit Representation of Frobenius Expansions

One can store a τ-adic expansion as a bitstring using Huffman encoding (see [54, pg 56–58]): for example coefficient 0 is represented as 0, coefficient 1 as 10 and coefficient -1 as 11. Hence a τ-adic expansion of length L and weight w requires $L + w$ bits (hence, we typically require $\approx 5L/3$ bits for a random τ-adic). We remark that with bounds given by Solinas [50] and an elementary counting argument, one can check that generally shorter representations are not possible, except by a fixed, small, additive constant.

However, if a τ-NAF is used then this can be shortened to $L + 1$ bits. For example, we represent coefficient 0 as 0, pair 01 as 01 and pair $0\bar{1}$ as 11, where "$\bar{1}$" means "-1".

For instance, the length 9 τ-NAF expansion $\tau^8 - \tau^6 - \tau^4 + 1$ (i.e., $[010\bar{1}0\bar{1}0001]$) could be represented as [0111110001].

4.3 Adding and Multiplying τ-adics

We will see in Chapter 6 an application of the use of Frobenius expansions with the GPS identification scheme [26, 43]. In that protocol we will need to add and multiply Frobenius expansions. The subtlety is that we require the result to have coefficients only in $\{-1, 0, 1\}$. Any coefficients which are outside this set must be reduced using the relation $2 = t\tau - \tau^2$, and this can lead to an increase in the degree of the expansion. There are two options for performing arithmetic:

1. Compute with coefficients in $\mathbb{Z}$ and reduce to coefficients in $\{-1, 0, 1\}$ at the end;
2. Perform the basic arithmetic operations so that all values have coefficients in $\{-1, 0, 1\}$ at all times.

Algorithm 4 presents an efficient addition algorithm of the second type. Note that this algorithm can also be used to convert any given τ-adic expansion with arbitrary integral coefficients into τ-adic expansions with coefficients in $\{-1, 0, 1\}$ when we add zero to the input (and compare with the algorithm proposed by Solinas, in that any given τ-adic expansion with arbitrary integral coefficients is converted into a τ-NAF expansion).

The important feature of this algorithm is that, although the carry values c_0, c_1 are integers, the arrays a, b and d only ever contain entries in the set $\{-1, 0, 1\}$. Hence this algorithm may be suitable for implementation on smart cards.

We remark that multiplication of τ-adics expansions can be done by treating the expansions as polynomials and performing a polynomial multiplication and then reducing the final result to coefficients in $\{-1, 0, 1\}$. However, one must take into account the sizes of the integer coefficients, which can be very large when we use τ-adic expansions of large degree. Hence, it may be necessary large memory space to store coefficients of such size.

An alternative to perform multiplication of τ-adics is to do repeated calls to Algorithm 4. This option, although we did not evaluate the impact of repeated shift-add to the performance protocols, seems to be the most suitable for restricted devices, like smart cards.

4.3.1 The Optional Randomisation Step

The optional randomisation step in line 12 (Algorithm 4) will be needed in the protocol (see Chapter 6 for details). It is the following: if i is less than some parameter $\mathcal{K}'$ then generate a uniform random $b \in \{-1, 0, 1\}$ and arrange that $x = b$ by adding $(b - x)(\tau^i - \tau^{m+i})$. This corresponds to taking an equivalent representation for the sum. Note that this operation may increase the degree of the result by m and requires storing a 'carry' to be

Algorithm 4 Efficient τ-adic addition

System Parameters: Frobenius polynomial $\tau^2 - t\tau + 2$ where $t = \pm 1$.
Input: τ-adic expansions a and b.
Output: $d \equiv (a + b)$.

```
1: deg = max(deg($a$), deg($b$))
2: $c_0 = 0$, $c_1 = 0$, $i = 0$ and $d = 0$
3: while ($i \leqslant$ deg) or ($c_0 \neq 0$) or ($c_1 \neq 0$) do
4:     $x = a[i] + b[i] + c_0$
5:     $c_0 = c_1$ and $c_1 = 0$
6:     if ($x < -1$) or ($x > 1$) then
7:         $q = \mathrm{sign}(x) \times (|x|\ div\ 2)$
8:         $x = x - 2q$
9:         $c_0 = c_0 + tq$
10:        $c_1 = c_1 - q$
11:    end if
12:    [Optional randomisation step]
13:    $d[i] = x$
14:    $i = i + 1$
15: end while
16: Return $d$
```

included when computing the $(m+i)$-th term[1]. Note that the randomisation method of Ebeid and Hasan [21] does not give uniform output of the low-degree coefficients and so it is not sufficient for our protocol.

Even with the non-randomised version, one sees that the degree can increase significantly if the carry values c_0, c_1 take nonzero values of large enough absolute value. Indeed, from Algorithm 4 one can derive a non-deterministic method to construct, given a τ-adic a, a τ-adic b such that $\deg(a + b) > \deg(b) \geqslant \deg(a)$ (note that the difference $\deg(a + b) - \deg(b)$ is bounded for fixed a). For example, given $a = 1 + \tau^3 - \tau^5$ one can choose $b = -\tau^5 + \tau^7$ so that the result of computing $a + b$ using Algorithm 4 is $1 + \tau^3 - t\tau^6 + t\tau^8 - \tau^9$. Indeed, it is clear that one can choose the first $\deg(a)$ coefficients of b at random, so there are at least $3^{\deg(a)}$ such choices

[1]We are using the polynomial $1 - \tau^m$, but could use any polynomial equivalent to zero and with constant coefficient 1.

for b (due to the choices available when constructing b there are many more choices for b, see heuristic 1 below). Similarly, one can choose b such that $\deg(a+b) < \deg(a)$ (for example, for the above a take $b = -\tau^3 + \tau^5$ so that $a + b = 1$); again there are about $3^{\deg(a)}$ possible choices.

4.4 Arithmetic Required for τ-GPS Protocol

Our variant of the GPS protocol will require computing $y = r + sc$ where r, c and s are all Frobenius expansions. The multiplication $s \times c$ can be performed by repeated shifting and addition using Algorithm 1 (possibly combined with a Karatsuba[2] approach). Note that the arithmetic coming from Algorithm 4 is not associative (for example $(1+1)+-1 \neq 1+(1+-1)$) though all results are equivalent. There are several natural algorithms for computing sc and, due to lack of associativity, they will generally give different results. For the analysis of the protocol in Section 6.6 we assume that a fixed algorithm is used for computing sc in the protocol and in the proof of Theorem 6.6.2.

Clearly, the product sc can have degree larger than $\deg(s)+\deg(c)$ but in practice it is not much bigger. We have performed a number of experiments using MAGMA [10] in order to check the degree of sc for random s and c of certain size. We define $\mathcal{C}$, $\mathcal{S}$ and $\mathcal{R}$ respectively the maximum lengths for the challenge, the secret key and the commitment source. We will see in Section 6.6 that we need $\mathcal{C} \geqslant 32$ such that the challenge cannot be easily guessed by the prover and we need $\mathcal{S} \geqslant 150$ such that the private key be resistant to DLP algorithms. Therefore, taking $(\mathcal{C}, \mathcal{S}) = (32, 150)$ and

[2]Divide-and-conquer algorithm discovered by A. Karatsuba in 1960, used for quickly multiplying large n-digit numbers. The time complexity is $O(n^{log_2 3})$, which is faster than the classical $O(n^2)$.

randomly chosen $c \in \mathcal{T}_{\mathcal{C}}$ and $s \in \mathcal{T}_{\mathcal{S}}$, we could observe that after 2^{40} random pairs (s, c) we never found $\deg(s \times c) > \deg(s) + \deg(c) + 5$.

Definition 22. *For $\mathcal{S}, \mathcal{C} \in \mathbb{N}$, we define*

$$\Phi = (\mathcal{S} - 1) + (\mathcal{C} - 1) + 5 = \mathcal{S} + \mathcal{C} + 3$$

and

$$\mathcal{T}_{\Phi,\mathcal{R}} = \Big\{ y \in \mathcal{T}_{\mathcal{R}} : y_n \neq 0 \textit{ for some } n \textit{ such that } \Phi \leqslant n < \mathcal{R} \Big\}.$$

Our protocol will compute $r + sc$ where $\deg(r)$ is much bigger than $\deg(s) + \deg(c)$ and it is important to ensure that the degree is not likely to increase. For a given choice of sc of degree Φ there seem to be $\leqslant 3^{\Phi} \times 2^{\mathcal{K}-1}$ values for r satisfying $\deg(r) < \Phi + \mathcal{K}$ and $\deg(r + sc) \geqslant \Phi + \mathcal{K}$. Hence, the probability that $\deg(r + sc) \geqslant \Phi + \mathcal{K}$ can be estimated as

$$\frac{3^{\Phi} 2^{\mathcal{K}-1}}{3^{\Phi+\mathcal{K}}} = \frac{2^{\mathcal{K}-1}}{3^{\mathcal{K}}} = \frac{1}{3}\left(\frac{2}{3}\right)^{\mathcal{K}-1} = \frac{1}{3(3/2)^{\mathcal{K}-1}} = \frac{1}{3^{1+(\mathcal{K}-1)\log_3(3/2)}} \approx \frac{1}{3^{0.63+0.37\mathcal{K}}}.$$

If $\mathcal{K}$ is sufficiently large then the probability of this event for randomly chosen r is negligible. Our experiments showed that for $(\mathcal{C}, \mathcal{S}, \mathcal{K}) = (32, 150, 20)$, the probability of $\deg(r + sc) \neq \deg(r)$ over random $c \in \mathcal{T}_{\mathcal{C}}$, $s \in T_{\mathcal{S}}$ and $r \in \mathcal{T}_{\Phi+\mathcal{K}}$ is approximately $0.00004 \leqslant 1/3^{0.37 \times 20 + 0.63} \approx 0.0001$. For $(\mathcal{C}, \mathcal{S}, \mathcal{K}) = (32, 150, 50)$ we never found an example with $\deg(r + sc) \neq \deg(r)$.

The security proof will require a stronger statement about the probability of $\deg(r + sc)$ being large for some choice of $c \in \mathcal{T}_C$. We have performed experiments and the results seem to be comparable to the previous case. Hence we propose the following heuristic. We assume that *negligible* is defined according to Definition 6.

Heuristic 1. *Fix $s \in \mathcal{T}_{\mathcal{S}}$. Suppose $\mathcal{K} \in \mathbb{N}$ is sufficiently large and let $\mathcal{R} = \Phi + \mathcal{K}$. Then the probability over $r \in \mathcal{T}_{\mathcal{R}}$ that there exists some $c \in \mathcal{T}_{\mathcal{C}}$ such that* $\deg(r + sc) \geqslant \mathcal{R}$ *is negligible. More precisely,*

$$\#\{r \in \mathcal{T}_{\mathcal{R}} : \exists\ c \in \mathcal{T}_{\mathcal{C}} \text{ with } \deg(r + sc) \geqslant \mathcal{R}\} = O(3^{\Phi}2^{\mathcal{K}}) = \tilde{O}(3^{\Phi+0.63\mathcal{K}}).$$

We also need to know how likely it is that $\deg(r + sc) \leqslant \Phi$. The above arguments indicate there should be at most 3^{Φ} such choices for r. Thus, we can state the following heuristic:

Heuristic 2. *Fix $s \in \mathcal{T}_{\mathcal{S}}$. Suppose $\mathcal{K} \in \mathbb{N}$ is sufficiently large and let $\mathcal{R} = \Phi + \mathcal{K}$ Then the probability over $r \in \mathcal{T}_{\mathcal{R}}$ that there exists some $c \in \mathcal{T}_{\mathcal{C}}$ such that* $\deg(r + sc) < \Phi$ *is negligible. More precisely,*

$$\#\{r \in \mathcal{T}_{\mathcal{R}} : \exists\ c \in \mathcal{T}_{\mathcal{C}} \text{ with } \deg(r + sc) < \Phi\} = O(3^{\Phi}).$$

A further subtlety of our basic (i.e., without randomisation) addition algorithm is that there is not necessarily a unique τ-adic solution x to the equation $a + x = b$. For example, it is easy to check that $-1 + x = 1 + \tau$ has no solution in $\mathcal{T}_3$. Similarly, one can check that $-1 + x = -\tau + \tau^2$ has the two (necessarily equivalent) solutions $x = 1 - \tau + \tau^2$ and $x = -1$ when $\tau^2 - \tau + 2 = 0$. The extra randomisation in line 12 of Algorithm 4 is included precisely to 'smooth out' this issue. In particular, it greatly reduces the probability that an equation of the form $a + x = b$ cannot be solved in the important case when $\deg(a) < \deg(b)$.

4.5 Summary

In this chapter we revised the main properties of Koblitz curves and Frobenius expansions (a.k.a. τ-adic expansions). We remarked that the most popular choice for Koblitz curves are curves over $\mathbb{F}_{2^m}$ for some prime $m > 1$. We saw that if an integer n is replaced by its equivalent Frobenius expansion, the point multiplication $[n]P$ can be performed, in practice, approximately 4 times faster than the standard double-and-add scalar multiplication, or about 50% faster than any previous known point multiplication method.

We used the fact that Frobenius expansions are not unique, i.e., if we look for expansions of degree less than some bound, we find many different (although equivalent) expansions which represent the same integer modulo $\mathbf{r}$, where $\mathbf{r}$ is the order of P. We also stated that in case we need a unique representation for a given equivalence class of integer, we can use a τ-NAF expansion of degree less than m.

Furthermore, we explored the property that a τ-adic can be mapped to $x + y\tau$ for integers x, y and we proved bounds for x and y. We also analysed the distribution of random τ-adics and random τ-NAFs and conjectured that we can shorten the bounds for x and y based on the distribution.

Finally, we presented the non canonical arithmetic of Frobenius expansions, specially when we require that all coefficients be in $\{-1, 0, 1\}$. We checked that the arithmetic is not associative, and when we add τ-adics a and b, we can get many different results (e.g. different degrees for the resultant expansion), but all of them are equivalent. We also proposed a randomisation step in the algorithm used to add and multiply τ-adics in order to avoid a statistical attack in the τ-GPS protocol.

Chapter 5

The Frobenius Expansions DLP

Contents

In this chapter we define three variants of the Frobenius expansion DLP and present algorithms to solve each of them.

We begin the chapter introducing three new computational problems, namely, the general τ-DLP, the τ-NAF DLP and the weight-w τ-DLP. Then we concentrate on low-memory algorithms, and propose the use of the Gaudry-Schost algorithm for solving the τ-DLP. We analyse this algorithm and also fill some gaps in its previous analysis.

5.1 Motivation

We recall that the elliptic curve discrete logarithm problem (ECDLP) is: given $P, Q \in E(\mathbb{F}_{q^m})[\mathbf{r}]$ to find $n \in \mathbb{Z}/\mathbf{r}\mathbb{Z}$ such that $Q = [n]P$. We discussed in Chapter 4 that one can represent n as a Frobenius expansion $n(\tau)$ in order to perform point multiplications more efficiently. We also recall from Section 4.2.4 that a Frobenius expansion $n(\tau) = n_0 + n_1\tau + n_2\tau^2 + \ldots + n_L\tau^L$ can be mapped to $x + y\tau$, for integers x, y of certain sizes. Evidently, to solve the ECDLP of Q to the base P it is sufficient to compute either the integer $n \in \mathbb{Z}/\mathbf{r}\mathbb{Z}$ or the coefficients n_j of a Frobenius expansion of n or alternatively the integers x and y. The main aim of the chapter is to present algorithms to solve the ECDLP when n is a Frobenius expansion of length L. Under the assumption that the standard DLP (i.e., computing the integer n) is hard, we will show that for some cases it is better to compute the coefficients n_j and for other cases it is better to compute x and y. We are particularly interested in expansions of relatively small length L, i.e. $L < m$.

The motivation behind this research is two-fold. The first motivation is to analyse the security of elliptic curve cryptosystems. Solinas [51], following an idea of H. Lenstra, suggested that rather than choosing a random integer n and then converting to a Frobenius expansion $n(\tau)$, in certain cryptosystems it might be more efficient to generate a random Frobenius expansion directly. The temptation then is to choose a relatively short and/or sparse value for $n(\tau)$. If this is done then we must re-evaluate the difficulty of the discrete logarithm problem (and other computational problems). A further issue is that the existing security proofs may not directly apply (see Lange and Shparlinski [35] for results on the uniformity of distribution of the resulting points). For some systems it may be necessary to develop bespoke security proofs for the Frobenius expansion case (see Chapter 6 and [8] for an example of this).

The suggestion of Solinas leads to the following problem: determine the most efficient way to generate ECDLP instances $Q = [n]P$ of a given difficulty. For example, one could choose random integers n of a certain bitlength, or random Frobenius expansions of a certain degree, or random sparse Frobenius expansions of a certain degree, etc. We do not attempt to answer this question (the solution will depend delicately on the relative costs of doubling, addition and q-powering; which in turn depend on the finite field in question, the curve equation, and what sort of coordinate system is being used). However, it is clear that a necessary prerequisite to answering this question is a study of algorithms to solve the DLP in all these situations. Note that one approach to this problem is using product exponents, see [14, 19, 32]. Also see [18] for a solution if P is fixed and sufficient memory is available.

The second motivation is more philosophical. Recall that there are numerous computational problems which admit exhaustive search algorithms in time $\tilde{O}(N)$ (for some measure N of the size of the problem) and time/ memory tradeoff algorithms in time and space $\tilde{O}(\sqrt{N})$. Van Oorschot and

Wiener [57] show that such problems can also be solved with low memory randomised methods (called parallel collision search) in expected time $\tilde{O}(N^{3/4})$. For distributed computing using $\mathbf{P}$ processors the above complexities generically become $\tilde{O}(N/\mathbf{P}), \tilde{O}(\sqrt{N}/\mathbf{P})$ and $\tilde{O}(N^{3/4}/\mathbf{P})$ respectively.

For some of the above problems (e.g., the discrete logarithm problem) there also exist low memory randomised algorithms (Pollard-rho and kangaroo [41]) which run in expected time $\tilde{O}(\sqrt{N})$ (respectively $\tilde{O}(\sqrt{N}/\mathbf{P})$, see [58]). It is natural to ask whether every time/memory tradeoff algorithm has a randomised low-memory counterpart with the same asymptotic complexity (possibly with worse constants). There are examples of computational problems with a good time/memory tradeoff algorithm (i.e., time and space complexity $\tilde{O}(\sqrt{N})$ for problem space of N elements) but for which there is no known low-memory algorithm with complexity $\tilde{O}(\sqrt{N})$ (for example, the low Hamming weight DLP [53], small CRT RSA private exponents [44] and the product DLP [14, 19]).

In this chapter, we divide the Frobenius expansion DLP into three computational problems, namely the general τ-DLP, the τ-NAF DLP and the weight-w τ-DLP (see Section 5.2). For the first two, it is natural to use an efficient time/memory tradeoff algorithm to solve them, and we use Gaudry-Schost [25] 2-dimensional approach to transform it into a low-memory randomised algorithm. For the the weight w τ-DLP, it can also be efficiently solved by a time/memory tradeoff algorithm, but we do not know any low-memory algorithm with complexity better than van Oorschot and Wiener's $\tilde{O}(N^{3/4})$.

5.2 The Frobenius Expansion DLP

Given $P, Q \in E(\mathbb{F}_{q^m})$ the standard discrete logarithm problem is to find $n \in \mathbb{Z}$ (if it exists) such that $Q = [n]P$. In the most common case, using Koblitz curves, it is enough to use $m = 163$ for the standard DLP to be intractable. The distributed Pollard-rho method using equivalence classes in $E(\mathbb{F}_{2^{163}})$ needs $\approx \sqrt{\frac{\pi 2^{163}}{4(163)}} \approx 2^{77.6}$ group operations.

We now introduce the problems of interest in this chapter. In order to construct more efficient schemes for restricted environments, we need to use Frobenius expansions of length not chosen in the range $[0, m]$, but in $[0, L]$, where usually $L < m$.

Definition 23. ***General*** τ***-DLP***. *Given inputs $P, Q \in E(\mathbb{F}_{q^m})[\mathbf{r}]$, $L \in \mathbb{N}$, find $n(\tau) \in \mathcal{T}_L$ (if it exists) such that $Q = [n(\tau)]P$.*

Definition 24. τ***-NAF DLP***. *Given inputs $P, Q \in E(\mathbb{F}_{q^m})[\mathbf{r}]$, $L \in \mathbb{N}$, find a τ-NAF $n(\tau)$ of length L (if it exists) such that $Q = [n(\tau)]P$.*

Definition 25. ***Weight*** w τ***-DLP*** *. Given inputs $P, Q \in E(\mathbb{F}_{q^m})[\mathbf{r}]$, $L, w \in \mathbb{N}$, find $n(\tau) \in \mathcal{T}_L$ of weight w (if it exists) such that $Q = [n(\tau)]P$. We are particularly interested in small w.*

Remark 1: We stress that $n(\tau)$ having low degree does not imply that $n \equiv n(\lambda) \pmod{\mathbf{r}}$ is a small integer (remember, λ is the eigenvalue of Frobenius, see Definition 21). Hence the above problems are not special cases of the problem of solving the discrete logarithm problem in a short interval.

Remark 2: Although the τ-NAF and the weight-w τ-DLP are related (we expect that both have a small number of nonzeros, but note that the weight bound on the τ-NAF is proportional to the length of the expansion and on the weight-w DLP the bound is fixed), here we state the latter as a separate problem (note that n can have a low Hamming weight, e.g., $n = 1000000000110000000001$, but not necessarily, n is a NAF) because there are some attacks which can be applied when n has low Hamming weight, regardless whether n is a τ-NAF or not.

Remark 3: The goal is to have a special algorithm for the low Hamming weight DLP which performs better than general discrete log algorithms. However, from some value of w there is a crossover when it is better to use a general algorithm.

5.3 Solving the τ-DLP

We now consider the standard methods for solving the DLP (exhaustive search, baby-step-giant-step) in our new setting. Let $\mathcal{C}$ be the coefficient set for the τ-adic expansions (we always assume $0 \in \mathcal{C}$) and let c be the number of elements of $\mathcal{C}$ (i.e., $c = \#\mathcal{C}$). We analyse each method for a general coefficient set $\mathcal{C}$ and then we give some examples when $c = 3$ (i.e. $\mathcal{C} = \{-1, 0, 1\}$), since this is the most popular choice.

5.3.1 Exhaustive Search

An obvious method to solve the τ-DLP is by exhaustive search.

5.3.1.1 General τ-DLP.

When solving the general τ-DLP, we need $O\big(c^L\big)$ group operations to find n. Note that, when $\mathcal{C} = \{-1, 0, 1\}$, we need to exhaustive search over 3^L possibilities, although we have showed in Section 4.2.2 that the number of equivalence classes is only $\tilde{O}(2^L)$.

However, it is possible to solve the general τ-DLP in $\tilde{O}(2^L)$ by using the fact that every τ-adic expansion can be mapped down to $x + y\tau$, for $x, y \in \mathbb{Z}$ and $|x| < a = 2.81\sqrt{2^L}$, $|y| < b = 2\sqrt{2^L}$ (see Theorem 4.2.2). Using exhaustive search over x and y, we need $< 22.48(2^L)$ group operations.

Furthermore, if we use the results of the τ-adics distribution (see Conjecture 1, Section 4.2.6) then we have $a = 2.28\sqrt{2^L}$ and $b = 1.8\sqrt{2^L}$ and therefore, the exhaustive search needs $< 16.42(2^L)$ group operations.

5.3.1.2 weight w τ-DLP.

We must list all Frobenius expansions of length L and weight w. To do this one first generates all binary strings of weight w and length L. This can be efficiently done by using Algorithm 5 below. Once all binary strings are generated, to get a τ-adic expansion one replaces the w ones in the binary number by every combination of the nonzero elements of $\mathcal{C}$. The number of such expansions is clearly $(c-1)^w\binom{L}{w}$.

Therefore, the exhaustive search algorithm for the weight w τ-DLP requires $\tilde{O}((c-1)^w\binom{L}{w})$ group operations.

Table 5.1 shows the number of group operations for $L = 160$ and some

Algorithm 5 Generating all binary strings of weight w and length L

```
Input: length $L$, weight $w$
Output: all binary strings of weight $w$ and length $L$
1: Initialise a length $L$ array with all bits 1 in the rightmost $w$ positions;
2: output array;
3: repeat
4:    $i \leftarrow$ position of leftmost bit 1;
5:    if $i$ is not leftmost position then
6:       Move leftmost bit one position to left;
7:    else
8:       $j \leftarrow$ position of leftmost bit 1 which can be moved to left;
9:       Move bit at position $j$ one position to left and bring all 1 bits to
         the left of $j$ to the positions immediately before $j$;
10:   end if
11:   output array;
12: until all bits 1 in the first $w$ positions of the array
```

values of w. We can see that for $w \leqslant 38$ we can solve the DLP with fewer group operations than the exhaustive search for the general case.

Table 5.1: Number of group operations for $L = 160$ and weight w.

w	Group Operations
5	$2^{34.61}$
10	$2^{61.01}$
20	$2^{103.57}$
30	$2^{137.76}$
37	$2^{158.09}$
38	$2^{160.78}$
39	$2^{163.43}$

5.3.1.3 τ-NAF DLP.

If one wants to exhaustive search over τ-NAFs, she needs to know how to generate all τ-NAFs of length $< L$. This can be efficiently done using Algorithm 5 to obtain all binary expressions of weight w and length $L' = L - w$, for $w = 1$ to $\lceil L/2 \rceil$. We then replace each bit '1' in the binary expression with a pair '$0x$' of elements in $\mathcal{C}$ where x is a nonzero element of $\mathcal{C}$. Finally, we discard the leftmost bit of each resultant expression (which is always zero when expressions are generated that way).

We remember from Section 4.2.1.1 that the number of τ-NAF expansions with degree less than L is $N_L = \frac{4}{3}2^L - \frac{1}{3}(-1)^L = \frac{4}{3}2^L + O(1)$. Notice that this formula only holds for a coefficient set $\mathcal{C} = \{-1, 0, 1\}$. Theorem 5.3.1 presents a similar formulae for a general coefficient set $\mathcal{C}$. The proof is analogous to the proof of Theorem 4.2.1.

Theorem 5.3.1. *Let $L \in \mathbb{Z}$. Let $c \in \mathbb{N}$ be the number of elements of the coefficient set $\mathcal{C}$ and $c' = c - 1$. The number of possible τ-adic NAF expansions of degree less than L is*

$$N_L = c_1\left(\frac{1+\sqrt{1+4c'}}{2}\right)^L + c_2\left(\frac{1-\sqrt{1+4c'}}{2}\right)^L = \tilde{O}\left(\left(\frac{1+\sqrt{1+4c'}}{2}\right)^L\right) \tag{5.1}$$

for some $c_1, c_2 \in \mathbb{Q}$.

Note that, when $\mathcal{C} = \{-1, 0, 1\}$, it follows that

$$N_L = \frac{4}{3}2^L - \frac{1}{3}(-1)^L = \frac{4}{3}2^L + O(1).$$

From (5.1), the total number of τ-NAF is $N_L = O((\frac{1+\sqrt{1+4c'}}{2})^L)$ where $c' = c - 1$, so, one needs to compute N_L group operations to find the discrete log n. When $\mathcal{C} = \{-1, 0, 1\}$, we have a $\tilde{O}(2^L)$ algorithm to solve the τ-NAF DLP.

Note that if we use the fact that we can map a τ-adic to $x+y\tau$, for integers x, y and use the τ-NAF distribution (see Conjecture 2, Section 4.2.6.2), we can use exhaustive search over $x < a = 1.53\sqrt{2^L}$ and $y < b = 1.24\sqrt{2^L}$ and we end up with an algorithm which requires $\leqslant 7.59(2^L)$ group operations. We still have a $\tilde{O}(2^L)$ algorithm, but in this case, the constant implicit in the $\tilde{O}(\ )$ is bigger.

5.3.2 Baby-Step Giant-Step

One could also try to find n by using Shanks' baby-step giant-step (BSGS) algorithm [16]. Without loss of generality we assume that L is even.

5.3.2.1 General τ-DLP.

A naive way to solve the general τ-DLP using Shanks' baby-step giant-step would lead to an algorithm with running time and memory requirements $O\big(\sqrt{c^L}\big)$. Notice that when $c = 3$, we have complexity $O\big(\sqrt{3^L}\big)$, which is not optimal, since we have $O(2^L)$ equivalence classes. However, we can do better than this and find an algorithm with complexity $O(2^{L/2})$ as expected.

The idea is to try to find the discrete log as $x+y\tau$, for integers x, y of certain size. When $\mathcal{C} = \{-1, 0, 1\}$ we recall (see Theorem 4.2.2, Section 4.2.4) that there exist integers i, j where $|i| < a = 2.81\sqrt{2^L}$ and $|j| < b = 2\sqrt{2^L}$ such that $n(\tau) = i + j\tau$.

Therefore, one computes a sorted list $\mathcal{L}$ (the *baby steps*) which consists of pairs $\{([j]\tau(P), j) \mid j \in [-2\sqrt{2^L} \dots 2\sqrt{2^L}]$ and then computes the *giant steps*, which consist of pairs x, $\{(Q - [i]P, i) \mid i \in [-2.81\sqrt{2^L} \dots 2.81\sqrt{2^L}]$ checking if $Q - [i]P$ appears in $\mathcal{L}$. When a match is found, the discrete log is solved, since $n = i + j\tau$. The running time and memory requirements are

$O(\sqrt{2^L})$. The constant hidden in the $O(\ )$ is 9.62 for the running time, since we require $4\sqrt{2^L}$ group operations to compute the baby steps and $5.62\sqrt{2^L}$ to compute the giant steps. For the memory requirements, the hidden constant is 4 since we require $4\sqrt{2^L}$ group elements to store the baby steps.

If we use the τ-adic distribution (see Conjecture 1), then $a = 2.28\sqrt{2^L}$ and $b = 1.8\sqrt{2^L}$ and the hidden constant becomes 8.16 for the running time and 3.6 for the memory requirements.

In the case of another coefficient set $\mathcal{C}$, once one has determined the bounds a and b, the same algorithm can be used. As always with baby-step-giant-step methods, if the memory is constrained then one can split the problem unevenly to give lower memory cost but larger time cost.

5.3.2.2 τ-NAF DLP.

We know there exist τ-NAFs i, j of length $L/2$ such that $n = i + \tau^{L/2}j$. Hence the natural algorithm follows. The complexity is twice as the exhaustive search algorithm for τ-NAFs of length $L/2$, since we need to compute $\frac{4}{3}2^{L/2}$ baby steps and approximately the same number of giant steps. So the required number of group operations is $\frac{8}{3}2^{L/2} \approx 2^{L/2+1.42}$. In addition, we need $\frac{4}{3}2^{L/2}$ group elements to store the baby steps.

We remark that over all choices for τ-NAF i, j some values $i + \tau^{L/2}j$ are not in non-adjacent form, but this does not seriously affect our analysis.

If we use the τ-NAF distribution and use BSGS over integers x, y, then we need to compute $3.06\sqrt{2^L}$ baby steps and $2.48\sqrt{2^L}$ giant steps, resulting in $5.54\sqrt{2^L} \approx 2^{L/2+2.47}$ group operations and store $2.48\sqrt{2^L}$ group elements.

Therefore, the hidden constants in the $O(\ )$ are smaller if we use the algorithm over τ-NAFs.

5.3.2.3 Weight-w τ-DLP.

It is straightforward to extend Coppersmith's baby-step-giant-step algorithm (see [38, 53]) for the low Hamming weight DLP to solve the weight-w τ-DLP . We recall that $c' = \#\mathcal{C} - 1 = c - 1$. Coppersmith's deterministic algorithm needs $\tilde{O}(L\binom{L/2}{w/2})$ group operations (according to [37], this is achieved by dividing the exponent into two equal pieces, each one with Hamming weight $w/2$). The randomised version of Coppersmith's algorithm needs $\tilde{O}(\sqrt{w}\binom{L/2}{w/2})$ group operations. Stinson's generalisation (see [53]) of Coppersmith's deterministic method leads to an algorithm requiring $O(w^{3/2}(\log L)\binom{L/2}{w/2})$ group operations.

For the weight-w τ-DLP , we replace the w ones in the binary number by every combination of the nonzero elements of $\mathcal{C}$. Therefore, the complexities for Coppersmith's deterministic algorithm, its randomised version and Stinson's generalisation are respectively $\tilde{O}((c')^{w/2}L\binom{L/2}{w/2})$, $\tilde{O}((c')^{w/2}\sqrt{w}\binom{L/2}{w/2})$ and $O((c')^{w/2}w^{3/2}(\log L)\binom{L/2}{w/2})$ group operations.

Table 5.2 shows the number of group operations for the Stinson's generalisation of Coppersmith's deterministic method, when we use $L = 160$, $c' = 2$ and some values of w.

We can see that for $w \leqslant 31$ we can solve the DLP with fewer group operations than the BSGS for the general case.

Table 5.2: Number of group operations for $L = 160$ and weight w using BSGS.

w	Group Operation
5	$2^{20.48}$
10	$2^{37.37}$
20	$2^{59.94}$
30	$2^{77.79}$
31	$2^{78.36}$
32	$2^{80.95}$

5.4 Low-Memory Algorithms for Weight-w τ-DLP

Let $N = (c-1)^w\binom{L}{w}$ be the number of τ-adic expansions of length L for $E(\mathbb{F}_{q^m})$ with coefficient set $\mathcal{C}$. We recall that the BSGS algorithm requires time and memory $O(\sqrt{N})$. However, if the adversary has a limited amount of memory available, the number of steps required to find the DLP must be increased. For example, if only w words of memory are available for the adversary, the number of steps required for BSGS becomes $O(N/w)$, with $O(N/w)$ memory accesses (if one can only store w group elements, then he does w baby steps and N/w giant steps. This gives an algorithm with storage w group elements and running time N/w group operations). Hence, only w times faster than the exhaustive search case. Ideally we would like a low-memory algorithm to solve the weight-w DLP in time $\tilde{O}(\sqrt{N})$, but finding such an algorithm in general remains an open problem.

Using the methods of van Oorschot and Wiener [57] one can use a meet-in-the-middle attack to obtain a low memory algorithm with complexity $\tilde{O}(N^{3/4})$ for the weight-w DLP, with $O(\sqrt{N})$ memory accesses. We briefly give the details of this method. We assume without loss of generality that L is even.

The starting point is the same as the time/memory tradeoff: Let $Q =$

$[n(\tau)]P$ where $n(\tau)$ is a τ-adic of length L. We write $n = i + \tau^{L/2}j$ where i and j are τ-adics of length $L/2$. We want to find a match $[i]P = Q - [j]\tau^{L/2}(P)$.

The first step of the method in [57] is to construct an easily computed function

$$g_1 : E(\mathbb{F}_{q^m}) \longrightarrow \mathcal{S}$$

where $\mathcal{S}$ is the set of all τ-adics of length $L/2$. We also need a function $g_2 : E(\mathbb{F}_{q^m}) \to \{0,1\}$. Define $f_0(n(\tau)) = [n(\tau)]P$ and $f_1(n(\tau)) = Q - [n(\tau)]\tau^{L/2}(P)$. Finally, we obtain

$$f : E(\mathbb{F}_{q^m}) \longrightarrow E(\mathbb{F}_{q^m})$$

as $f(R) = f_{g_2(R)}(g_1(R))$.

The method of [57] is to iterate f from randomly chosen starting points R until a distinguished point is hit, in which case the start and end points are stored. This process is repeated many times. Once collisions are detected one can traverse the walks to find the exact collision points. There will be many 'useless' collisions; these are inevitable since the map g_1 is from a big set to a much smaller set and so we can have a collision from

$$g_1(R_1) = g_1(R_2) \qquad \text{and} \qquad g_2(R_1) = g_2(R_2)$$

for points $R_1, R_2 \in E(\mathbb{F}_{q^m})$. Eventually one expects to find a useful collision arising from the desired relation $Q - [j]\tau^{L/2}(P) = [i]P$ (in other words, $f_0(i(\tau)) = f_1(j(\tau))$) and we are done.

According to [57], the number of steps is $7n^{3/2}/\sqrt{w}$ and the number of memory access are $9n$, where $n = 2^{w/2}\binom{L/2}{w/2} \approx \sqrt{N}$. Therefore, for the weight-w DLP, the number of steps is approximately $(c-1)^{w/2}\frac{7\binom{L/2}{w/2}^{3/2}}{\sqrt{w}}$ and we need to access the memory approximately $9((c-1)^{w/2}\binom{L/2}{w/2})$ times.

When $\mathcal{C} = \{-1, 0, 1\}$, since the number of τ-adics of length $L/2$ and weight $w/2$ is $O(2^{w/2}\binom{L/2}{w/2})$ the main result of [57] implies that the number of steps of this algorithm is $2^{w/2}\frac{7\binom{L/2}{w/2}^{3/2}}{\sqrt{w}}$. Taking $N = 2^w\binom{L}{w}$ we end up with $\tilde{O}(N^{3/4})$. The number of accesses to the memory is therefore $\tilde{O}(\sqrt{N})$.

Table 5.3 shows the number of group operations of van Oorschot and Wiener's generalisation of Coppersmith deterministic method, when we use $L = 160$, $c' = 2$ and some values of w.

Table 5.3: Number of group operations for $L = 160$ and weight w using van Oorschot and Wiener.

w	Group Operations
5	$2^{21.58}$
10	$2^{42.92}$
20	$2^{71.52}$
23	$2^{76.92}$
24	$2^{81.18}$

We can see that for $w \leqslant 23$ the method is better than the distributed Pollard-rho in the whole group $E(\mathbb{F}_{2^{163}})$.

5.5 The 2-Dimensional Discrete Log Problem

In this section, we use the Gaudry-Schost Algorithm [25] to find a discrete log using a 2-dimensional pseudorandom walk.

Definition 26. *Let $E(\mathbb{F}_{q^m})$ be an elliptic curve, let Q, P_1, P_2 be points of E*

and let a, b be integers of known size. The 2-dimensional discrete log problem *(2-dim DLP for short) is to find integers x, y such that $Q = [x]P_1 + [y]P_2$, for $|x| \leqslant a$ and $|y| \leqslant b$.*

5.5.1 The Sample Space

We need to define the sample space in which the discrete log is supposed to lie. By assumption the discrete log lies within a rectangular box of sizes $2a$ and $2b$, centered in the cartesian origin $(0, 0)$. Therefore, we can define our sample space:

Definition 27. *Let P_1 and P_2 be points of an elliptic curve $E(\mathbb{F}_{q^m})$. Let a, b be integers of some known value. We define:*

$$\mathcal{X} = \big\{[x]P_1 + [y]P_2, \textit{such that } |x| \leqslant a, |y| \leqslant b\big\}.$$

We assume that there do not exist distinct (x_1, y_1) and (x_2, y_2) in $\mathcal{X}$ such that $[x_1]P_1 + [y_1]P_2 = [x_2]P_1 + [y_2]P_2$.

The number of elements of $\mathcal{X}$ is the area of the box in which the discrete log is supposed to lie. In other words:

$$\#\mathcal{X} = (2a).(2b) = 4ab.$$

5.5.2 Gaudry-Schost Low-Memory Algorithm

Gaudry and Schost [25] propose an algorithm for the 2-dim DLP. They define a pseudorandom walk in the set $\mathcal{X}$ of the form $R_{i+1} = R_i + [s_{\phi(R_i)}]P_1 + P_2$

(for more details, see Section 5.5.3).

They estimate the running time using simplifying assumptions. For instance:

- they assume that the pseudorandom walk stays inside $\mathcal{X}$;
- they do not consider that Tame-Tame and Wild-Wild collisions may occur;
- they also do not consider that some walks may not reach a distinguished point (an estimate of the probability that a walk does not reach a distinguished point is given in [58]);
- they do not give an analysis for the success probability.

We try to fill the gaps here, giving an estimate of how many steps are likely to be outside of the search box and analysing the success probability of the algorithm. We also conclude that "useless" collisions, i.e., Tame-Tame and Wild-Wild collisions do not seriously affect our analysis. As an application, we use Gaudry-Schost Algorithm to solve the general τ-DLP and the τ-NAF DLP.

We analyse the parallel case only, since that is the one used in practice. Basically, in our setting, we have $\mathbf{P}$ processors running in parallel. One half of them are tame kangaroos, each one starting from a random point and walking according to a 2-dimensional pseudorandom walk, to be defined later. The other half of the processors are wild kangaroos, each one starting from $Q+[W_1]P_1+[W_2]P_2$, for known integers W_1, W_2 and walking according to the same pseudorandom walk. We expect that after $\tilde{O}(\sqrt{\#\mathcal{X}})$ group operations (this is the total machine time, i.e., when we consider the total amount of work done by all processors), some tame kangaroo lands on one of the footprints of one of the wild kangaroos and then they continue following the same path until both land on a distinguished point and the DLP is solved. Now we give the details.

5.5.3 The Pseudorandom Walk

We recall from Section 3.3.2.1 that $\phi(R)$ maps the x-coordinate of a point R to an integer j, where j is computed modulo s (and s is the number of partitions of our group). Let s_j be a function to be defined later.

Each kangaroo starts at some random point $R_0 = [x_0]P_1 + [y_0]P_2$ and jumps according to the following pseudorandom walk:

x-direction: $x_{i+1} = x_i + s_{\phi(R_i)}$.

y-direction: $y_{i+1} = y_i + 1$.

So that $R_{i+1} = R_i + [s_{\phi(R_i)}]P_1 + P_2$. Notice that at each new point visited, we have $R_i = [x_i]P_1 + [y_i]P_2$. The walk stops when a distinguished point $R' = [x']P_1 + [y']P_2$ is reached (see Definition 13 for distinguished point). Then each R' is stored by a server, who coordinates the solution of the problem.

We remark that one advantage of choosing $+1$ in the y-direction is to avoid collisions within the same walk (self-collisions).

One should question that the walk is not close to random because in the y-direction the kangaroo always move one step to the North. We present the following heuristic:

Heuristic 3. *Let $\mathcal{X}$ be a rectangular box of sides $2a$ and $2b$. Run the above pseudorandom walk, starting from a point chosen uniformly at random, until a distinguished point is reached. Let $\mathcal{P}$ be the induced distribution on $\mathcal{X}$. We claim that the distribution of $\mathcal{P}$ over $\mathcal{X}$ is close to uniform.*

Note if all points are distinguished then this heuristic is trivially true. So, if the number of processors $\mathbf{P}$ is large then this heuristic is plausible.

We also give a picture where we plotted the walks of approximately $4,000$ kangaroos starting from random points in $(-a, a) \times (-b, b)$ and walking according to the pseudorandom walk defined above. Let $\mathcal{X}$ be our search area. We used $a = 98304$ and $b = 65536$ (these numbers arise in Section 5.6 with τ-adic expansions of length $L = 30$). Note that the area shown in the picture is larger than $\mathcal{X}$, so that we can see that some steps go outside the search area. In Section 5.5.6 we give an estimate for the number of steps outside $\mathcal{X}$. By observing Figure 5.1 we can note that the random walks give a good coverage of $\mathcal{X}$, which supports Heuristic 3.

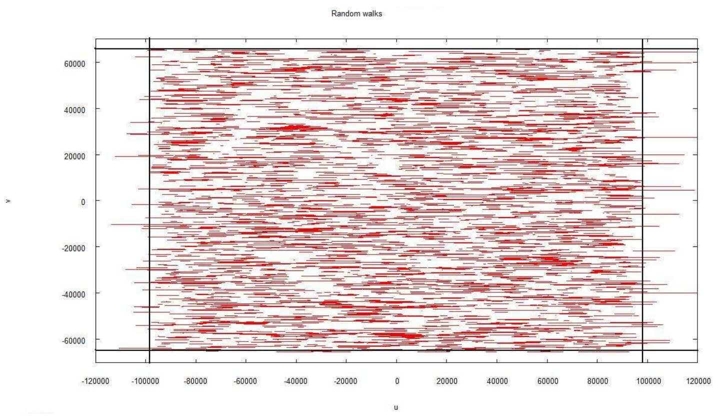

Figure 5.1: Distribution of $4,000$ random walks. The bounds for axis x and y are respectively $(-120000, 120000)$ and $(-70000, 70000)$. The black vertical lines represent $x = -a$ and $x = a$ for $a = 98304$ and the black horizontal lines represent $y = -b$ and $y = b$, for $b = 65536$.

5.5.4 The Number of Steps in Each Walk

We recall that each kangaroo will jump according to a 2-dimensional pseudorandom walk until a distinguished point is reached. By defining the distinguishing property, we can bound the number of steps in each walk. For example, according to [58], we can classify a point as "distinguished" if its binary representation has some fixed number of leading zeros. In our implementation, we classify a point as distinguished if its x-coordinate modulo some value θ is equal to zero. Let $1/\theta$ be the proportion of distinguished points. Therefore, we expect to reach a distinguished point after θ steps.

As a practical example, van Oorschot and Wiener [58] use $\frac{1}{\theta} = (0.93)2^{-32}$. They remark that such value is close to $(234/256)2^{-32}$, which can be achieved by defining a point as distinguished if it has 32 leading zero bits and the next 8 bits have a binary value less than 234.

It might happen that some walk never reaches a distinguished point. To avoid this, we must define an upper bound for the number of steps. If the kangaroo reaches that upper bound, we abandon the walk. This idea appears in [58]. Note that the work done on such walks is wasted for the algorithm since a collision on such points will never be detected. Therefore, we cannot count the steps of those walks in the analysis, but we must count them to compute the total number of steps of the algorithm. In Section 5.5.6 we give a theorem for the proportion of points that do not reach a distinguished point.

Notice that, depending on the position of the starting point and also on the size of each walk, some points can be outside the box, and therefore, outside our sample space. Of course such walks could be useful in practice, but for our analysis we need to use an argument about sampling n points from a set of size N, so we have to work within a fixed size region. Again, the steps of such walks must be counted to compute the total number of steps of the algorithm. In Section 5.5.6 we give an estimate for the number of steps outside $\mathcal{X}$.

5.5.5 Analysis

We recall from Chapter 3 that the success probability of Pollard-kangaroo is estimated using the mean step size. In the analysis of their 2-dimensional algorithm, Gaudry and Schost use the Birthday Paradox (see Section 3.3.1). In this section, we analyse Gaudry-Schost algorithm using probability arguments similar to the method used by Pollard to analyse the kangaroo method and give an estimate for the probability of success.

5.5.5.1 Choice of Parameters

In this Section, we present the parameters used in our implementation. We do not claim that they are the best choices, but the results obtained from MAGMA [10] experiments showed they are satisfactory (see Section 5.6.1).

- **P** (number of processors), **Q** (number of distinguished points) and θ (expected number of steps) - Note that Algorithm 6 is a **serial** implementation of Gaudry-Schost Algorithm, so our analysis is based on a scenario where **one** processor computes **Q** distinguished points, and it takes, on average, θ steps to reach a distinguished point. Hence, in Section 5.5.5.3 we consider the running time of Algorithm 6 to be $\mathbf{Q}\theta$.

In real life, the computation is performed in parallel and we usually do not have much control in the number of processors, but it really does not matter in practice. If we have few processors available, each of them runs lots of walks in the lifetime of the computation and computes a great number of distinguished points. In that case, if we need to compute $\mathbf{Q}$ distinguished points, each processor computes $\mathbf{Q}/\mathbf{P}$ of them and the running time will be $\mathbf{Q}\theta/\mathbf{P}$. Note that using more processors reduces the amount of work of each processor. Since the choice of $\mathbf{Q}$ and θ determines the running time, we leave that choice to be treated separately in Section 5.5.7.

Given the size of the problem and the amount of memory available, one determines the value of θ and therefore, t (see below). It is clear to see that the amount of storage depends on θ. If θ is relatively small, the walk is short, so there will be many distinguished points sent to the server. On the other hand, if θ is large, the walks are long and there will be less distinguished points sent to the server and therefore, less storage.

- t (parameter used to compute the length of steps in the x-direction) - We will see in Lemma 5.5.2 and Conjecture 3 that the expected maximum distance travelled from the origin is a function of θ and m. Since m depends on t, when t is very small, the mean absolute step size m will be small and therefore the maximum distance travelled will be small as well. As a result, we have very few steps outside the region but the walks will not be close to uniformly distributed over $\mathcal{X}$. On the other hand, if t is too large, we will have many steps outside the region. We conjecture that t should be $c_+ \log \theta$ for some constant c_+;

- s_i (steps in the x-direction) - We move the kangaroos in the x-direction using powers of 2. Here, unlike standard Pollard-kangaroo, we allow the kangaroos to move either to the East or to the West. In other words:

  $$s_i \in \{-1, 1, -2, 2, -4, 4, \ldots - 2^{\frac{t}{2}-1}, 2^{\frac{t}{2}-1}\};$$

  Note that for this case, the *mean absolute step size* is

  $$m = \frac{2^{\frac{t}{2}} - 1}{t/2}.$$

- Top (upper bound for number of steps) - It might happen that some walk never reaches a distinguished point. To avoid this, we must define an upper bound for the number of steps. If the kangaroo reaches that upper bound, we abandon the walk. This idea appears in [58]. We have chosen $Top = 20\theta$. See Section 5.5.6 for more details;

- T_1, T_2 (starting points for the tame kangaroos) - Each kangaroo starts from a random point inside $\mathcal{X}$. Trivially, if we start the tame kangaroos in $(-a, a) \times (-b, b)$, then every point in $\mathcal{X}$ can be potentially reached. However, we have many steps outside $\mathcal{X}$. We must define a bound such that we ensure that the kangaroos can potentially reach every point of $\mathcal{X}$ and also that we do not have too many steps outside $\mathcal{X}$. So, we start the tame kangaroos in $(-da, da) \times (-cb, cb)$, where c, d are positive real numbers less than 1. We define the values for c and d in Lemmas 5.5.1 and 5.5.3. In Section 5.5.6 we analyse the probability that steps go outside $\mathcal{X}$.

- W_1, W_2 (starting points for the wild kangaroos) - The wild kangaroos' starting points are in an region of sides $(-da, da) \times (-cb, cb)$, centered at Q. In other words, the wild kangaroos walk in the set $Q + \mathcal{X} = \{Q + P : P \in \mathcal{X}\}$. We will see in Section 5.5.5.4 that the area of interest for us will be the area where the wild kangaroos intersect the tame kangaroos. In Section 5.5.6 we give an estimate of the probability that wild steps go outside $Q + \mathcal{X}$.

Now we reduce the problem to a 1-dimensional random walk in $[0, b]$ in order to define the size of c.

Lemma 5.5.1. *Consider a pseudorandom walk which starts between points* 0 *and* b *(without loss of generality, we assume here that* b *is a positive integer). Let* $c \times b$ *for a positive real* $c < 1$ *be the most distant starting point from* 0*. Let* θ *be the number of steps and let the size of each step be* 1*. When* $c = 1 - \theta/b$*, every point in* $(0, b)$ *can be reached.*

Proof. Since each walk has length approximately θ, we need $c \times b + \theta$ to be at least equal to b to ensure that every point between $(0, b)$ can be reached. This leads to $c = 1 - \theta/b$. □

Now we reduce the problem to a 1-dimensional random walk in $[0, a]$. Before defining the size of d, we define the average distance from the origin.

Lemma 5.5.2. *Let* θ *be the number of steps. Let* O *be the starting point of some random walk. Let* m *be the mean step size, to the East or to the West. The* expected *distance from* O *after* θ *steps is* $m\sqrt{2\theta/\pi}$*.*

Proof. According to [39] (see also [59]), the average distance in absolute value from the origin in a 1-dimensional random walk when one takes θ steps of size 1 to the left or to the right is $D_{avr} = \sqrt{2\theta/\pi}$. When we abandon the restriction that the steps have size 1, it follows that the average distance reached from 0 to the left or to the right is $mD_{avr} = m\sqrt{2\theta/\pi}$, where m is the mean absolute step size. □

The expected *maximum* distance from some point is computed in [15]. They consider a random walk that starts at the origin and takes n steps uniformly distributed on the real interval $[-1, 1]$. Such value is $D_{max} \approx \sqrt{\frac{2n}{3\pi}}$. Now we use that result to state the following conjecture:

Conjecture 3. *Let θ be the number of steps. Let O be the starting point of some random walk. Let m be the mean step size, to the East or to the West. The* expected maximum *distance from O after θ steps is $Dist = \frac{2m}{\sqrt{3}}\sqrt{\frac{2\theta}{\pi}}$.*

Evidence for the conjecture: Let $D_{max} \approx \sqrt{\frac{2n}{3\pi}}$ as above. Note that $D_{max} = \frac{D_{avr}}{\sqrt{3}}$ for D_{avr} as in Lemma 5.5.2. Since to compute D_{max} we take steps distributed randomly on the real interval $[-1, 1]$, the mean absolute step size is 1/2. To get the mean absolute step size equals to m, we just multiply by $2m$. So it follows that $Dist = \frac{2m}{\sqrt{3}}\sqrt{\frac{2\theta}{\pi}}$.

Finally we define the value of d:

Lemma 5.5.3. *Consider a pseudorandom walk which starts between points 0 and a (again, without loss of generality, we assume that a is a positive integer). Let $d \times a$ for positive real $d < 1$ be the the most distant starting point from 0. let θ be the number of steps. Let t be parameter used to compute the length of the steps, let the size of each step be in $\{-1, 1, -2, 2, -4, 4, \ldots, 2^{\frac{t}{2}-1}, 2^{\frac{t}{2}-1}\}$. Let m be the mean absolute step size and let $Dist$ be the expected maximum distance reached from 0, according to*

Lemma 5.5.5.1. When $d = 1 - Dist/a$, *every point in* $(0, a)$ *can be reached with non-negligible probability.*

Proof. Note that even if the starting point was in the centre of $\mathcal{X}$ some walks could have length greater than a. However, those walks are very rare, because we need some special combination of positive and very large steps to reach some point beyond a. For example, if $a = 2^{17}$, $\theta = 2^5$ and the largest step size is 2^{13}, a walk which starts in the centre of $\mathcal{X}$ would have length $> a$ if all steps were equal to 2^{13}, since $(2^5)(2^{13}) = 2^{18} > a = 2^{17}$. If we choose d too far from a (in other words, too close from the centre of $\mathcal{X}$), although it is possible that points in a or very close to a can be reached, we will not have a distribution close to uniform as we assumed. By taking $d = 1 - Dist/a$ we ensure that most of the points which start in or close to d will reach a. Obviously, with such choice, some steps may go beyond a. See Section 5.5.6 for an estimate of the probability that steps go outside $\mathcal{X}$. □

5.5.5.2 Algorithm

Since we could not simulate a parallel implementation, Algorithm 6 presents a **serial** implementation of Gaudry-Schost Algorithm. Note that, in the parallel version, it does not make sense to compute first the tame steps, sort the list of distinguished points and then compute the wild steps and look for a match using binary search. In practice, we have all processors computing the tame and wild steps simultaneously and sending the information about distinguished points to the server. The server stores the distinguished points in a data structure which can be searched efficiently and when a Tame-Wild collision is found, we are done.

Remark 4: Notice that in Algorithm 6, we use the x-coordinates of the tame and the wild kangaroos to get an integer, because we need an integer to sort the list of distinguished points so that we can use Binary Search. As a result, it may happen that we have a collision $Tame = Wild$ but in fact, $x_{Tame} = x_{Wild}$ and $y_{Tame} = -y_{Wild}$. In this case, when we compute $n = (SD[index][2] - W_1) + (SD[index][3] - W_2)\tau$, we find the wrong value for the DLP. The solution for that is simple: Let $\mathbf{r}$ be the order of point P. Let $[c]P$ be the point where the tame kangaroo lands after its walk and let $[n+d]P$ be the point where the wild kangaroo lands after its walk (by definition, $[c]P$ and $[n+d]P$ are the same distinguished point). If Algorithm 6 outputs a solution which does not match with the DLP, we compute $c' = \mathbf{r} - c$. The solution for the DLP will be $n = c' - d$, since $[c']P = [n+d]P$;

5.5.5.3 Running Time

Recall that the DLP is solved when we find a Tame-Wild collision. We call this a *useful collision.* Note that Tame-Tame and Wild-Wild collisions do not solve the DLP. When computing the running time for a parallel counterpart of Algorithm 6, we need to count the number of steps needed to find a useful collision, considering the amount of work done by all processors in parallel. We call that the **total machine time**. Note that the running time for each processor individually it is just the total machine time divided by the number of processors, $\mathbf{P}$.

Lemma 5.5.4. *Let $\mathbf{P}$ be the number of processors, let $\mathbf{Q}$ be the (expected) total number of distinguished points that need to be computed such that a useful collision is found by the server and let $\frac{1}{\theta}$ be the proportion of distinguished points. The expected running time to solve the discrete logarithm*

Algorithm 6 Gaudry-Schost Algorithm - Serial version

```
Input: group $E(\mathbb{F}_q)$; points $P_1, P_2, Q \in E(\mathbb{F}_q)$; number of distinguished points $\mathbf{Q}$; proportion of distinguished points $1/\theta$; $Top$; $a$; $b$; $c \leftarrow 1 - \theta/b$; $d \leftarrow 1 - Dist/a$.
Output: integers $x$ and $y$ such that $Q = [x]P_1 + [y]P_2$.
1: Choose $t \in \mathbb{N}$
2: $PPi \leftarrow \{-1, 1, -2, 2, -4, 4, \ldots, 2^{\frac{t}{2}-1}, 2^{\frac{t}{2}-1}\}$
3: $PPj \leftarrow [1 : i \in [0 \ldots t-1]]$
4: $PP \leftarrow [PPi[i] * P_1 + PPj[i] * P_2 : i \in [0 \ldots t-1]]$
5: function PollardWalk (P, Pi, Pj)
6: for $length = 1$ to $Top$ do
7:    $u \leftarrow x_P \bmod t$
8:    $P \leftarrow P + PP[u+1]$
9:    $Pi \leftarrow Pi + PPi[u+1]$
10:   $Pj \leftarrow Pj + PPj[u+1]$
11:   if $x_P \equiv 0 \bmod \theta$ then
12:      Return $P, Pi, Pj$   /* Distinguished point found */
13:   end if
14: end for
15: end function
16: [Tame Kangaroo]
17: for $NrTame = 1$ to $\lceil \mathbf{Q}/2 \rceil$ do
18:    $T_1 \leftarrow Random(\lceil -d * a \rceil, \lceil d * a \rceil)$; $T_2 \leftarrow Random(\lceil -c * b \rceil, \lceil c * b \rceil)$
19:    $Tame \leftarrow [T_1]P_1 + [T_2]P_2$
20:    $Tame, T_1, T_2 \leftarrow PollardWalk(Tame, T_1, T_2)$
21:    [Constructing a list of distinguished points]
22:    if Tame is Distinguished then
23:       Store $[x_{Tame}, T_1, T_2]$ in a list $D$
24:    end if
25: end for
26: $SD \leftarrow Sort(D)$   /* Sorting List $D$ by its first component */
27: [Wild Kangaroo]
28: $W_1 \leftarrow 0$; $W_2 \leftarrow 0$
29: for $NrWild = 1$ to $\lceil \mathbf{Q}/2 \rceil$ do
30:    $Wild \leftarrow Q + [W_1]P_1 + [W_2]P_2$
31:    $Wild, W_1, W_2 \leftarrow PollardWalk(Wild, W_1, W_2)$
32:    if Wild is Distinguished then Binary Search $x_{Wild}$ in $SD$
33:    if $x_{Wild} = SD[index][1]$ then
34:       Output $x = (SD[index][2] - W_1)$, $y = \pm(SD[index][3] - W_2)$
35:    end if
36:    $W_1 \leftarrow Random(\lceil -d * a \rceil, \lceil d * a \rceil)$; $W_2 \leftarrow Random(\lceil -c * b \rceil, \lceil c * b \rceil)$
37: end for
```

problem using a parallel version of Algorithm 6 is $\mathbf{Q}\theta/\mathbf{P}$ *group operations.*

Proof. We have $\mathbf{P}/2$ processors representing tame kangaroos and $\mathbf{P}/2$ processors representing wild kangaroos. Each processor computes approximately $\mathbf{Q}/\mathbf{P}$ distinguished points. The expected length of a walk is θ, so the expected running time (for each processor) is $\mathbf{Q}\theta/\mathbf{P}$. □

Note that $\mathbf{Q}\theta/\mathbf{P}$ is the running time for the parallelised Gaudry-Schost Algorithm, not the running time of Algorithm 6. Clearly, the running time of Algorithm 6 is $\mathbf{Q}\theta$ group operations. Experimentally, we counted the total number of steps and we found all of them less or equal than $\mathbf{Q}\theta$. Obviously, when there is a collision, the running time is usually smaller than that.

5.5.5.4 Finding a collision

Now we need to know the number of steps needed for the server to find a useful collision. Hence, we need to compute the total machine time, considering the work done by all processors in parallel. In this section, we will define values k, k' and K. We assume that such values are number of steps considering all the work done, not only number of steps taken by one processor individually. So, to make things easier, we assume in this analysis that we have just one processor. Therefore, the final result (i.e., the value of K needed for the server to find a useful collision, see Theorem 5.5.7) represents the number of steps in the total machine time. When we use $\mathbf{P}$ processors running in parallel, each one will compute, on average, $K/\mathbf{P}$ steps.

Using Heuristic 3, we assumed that the points are randomly sampled from the sample space $\mathcal{X}$. We have approximately $(\mathbf{Q}/2)\theta$ points visited by tame kangaroos and approximately $(\mathbf{Q}/2)\theta$ points visited by wild kangaroos.

The starting points for the tame kangaroos are randomly chosen from $(-da, da) \times (-cb, cb)$ which means that the tame kangaroos nearly cover the whole area $\mathcal{X}$. Let us call T the region in which the tame kangaroos walk (i.e., $T = \mathcal{X}$). Obviously, it can happen that some tame kangaroos go outside T and also that some walks do not reach a distinguished point.

The first wild kangaroo starts at Q an then its starting points are shifted in an area of sides $(-da, da) \times (-cb, cb)$, centered in Q. Let us call W the region in which the wild kangaroos walk. Using the previous notation, $W = Q + \mathcal{X}$. Again, it can happen that some wild kangaroos go outside W and also that some walks do not reach a distinguished point.

We call a step *good* if it lies in T or in W and the end of its walk reaches a distinguished point. We call a step *bad* otherwise. In the rest of this chapter we use K to denote the total number of steps taken by kangaroos, so that $K \approx \mathbf{Q}\theta$. We let k be the number of good steps. Later we will use k' to denote the number of good steps which lie also in the region of overlap between the space of tame kangaroo walks and the space of wild kangaroo walks.

Notice that it can happen that some steps of a walk go outside the regions of interest (T or W) but afterwards the walk returns to the region and reaches a distinguished point inside T or W. In that case, only the steps that go outside T or W will be counted as bad steps. All steps of such a walk which are inside the region of interest will be counted as good steps.

We follow the approach of [25] and define $A = T \cap W$, i.e., the region where the tame kangaroos footprints can overlap the wild kangaroos. Let

k' be the total number of good steps in A (Tame or Wild). Assuming that the distribution of tame and wild kangaroos is close to uniform, we have approximately $k'/2$ tame kangaroo steps and approximately $k'/2$ wild kangaroo steps in A.

5.5.5.5 The Probability of Success

In this section we give an estimate for the probability of success of the Algorithm in terms of the number of steps. This enables us to determine the total machine time for any desired success probability. Note that until the server detects a Tame-Wild collision, some Tame-Tame and Wild-Wild collisions may occur. For the analysis of the success probability of the algorithm, we do not mind whether such "useless" collisions occur. We just want to be sure that *there is* a Tame-Wild collision.

Let S_1 be the set of points visited by tame kangaroos which are in the region where tame and wild kangaroos overlap (i.e., A). Perhaps there are some Tame-Tame collisions, but not very many. For the parameters we are considering we expect only one or two Tame-Tame collisions. Hence $\#S_1 \approx k'/2$. Now let S_2 be the set of points visited by wild kangaroos in the region of overlap. Again, $\#S_2 \approx k'/2$.

Finally, what is the probability that S_1 and S_2 are disjoint (i.e., there are no collisions between them)? If we think of choosing S_1 first and then choosing S_2 then each element in S_2 has to avoid all the elements in S_1. So the probability that S_2 is disjoint from S_1 is

$$P_1 \approx \left(1 - \frac{k'/2}{\#A}\right)^{k'/2} \approx e^{-(k'/2)^2/(\#A)} \approx e^{-(k')^2/(4\#A)}$$

Hence, the probability that *there is* a Tame-Wild collision is

$$1 - e^{-(k')^2/(4\#A)}. \tag{5.2}$$

To get success probability 0.5 in the worst case (i.e., $\#A = \frac{1}{4}(\#\mathcal{X})$) we solve $1 - e^{-(k')^2/(4\#A)} = 0.5$ and so

$$k' = \sqrt{(\#\mathcal{X}) \ln 2}.$$

5.5.6 Counting *bad* steps

In this section we deal with points that go outside the bounds of T and W and walks that do not reach a distinguished point. Both cases are not counted in the probability argument, but we need to add them to compute the total number of steps.

Definition 28. *We call N_1 the total number of steps abandoned, due to the walks that do not reach a distinguished point.*

Definition 29. *Let T be the region where the tame kangaroos walk and let W be the regions where the wild kangaroos walk. We call N_2 the total number of steps that go outside T or W.*

5.5.6.1 Walks That Do Not Reach a Distinguished Point

We quote a Theorem from [58] that shows the proportion of steps that do not reach a distinguished point. The proof of this theorem can also be found in [58].

Lemma 5.5.5. *Let K be the total number of steps. Let Top be a parameter which determines the maximum walk length. For $Top = 20\theta$, we have $N_1 < (5 \times 10^{-8})K$.*

Proof. The maximum walk length is 20θ and we abandon any walk which exceeds the maximum length. Since the probability of a distinguished point to be reached is $1/\theta$, the probability that a distinguished point *will not* be reached is $1 - 1/\theta$.

After walking up to the maximum walk length the probability that some walk length exceeds 20θ is $(1 - 1/\theta)^{20\theta} \approx e^{-20}$. Each abandoned walk has length about 20 times longer than the average θ, therefore, the proportion of points that do not reach a distinguished point is $\approx 20e^{-20} < 5 \times 10^{-8}$.

If K is the total number of steps, it follows that

$$N_1 < (5 \times 10^{-8})K.$$

□

5.5.6.2 Footsteps Outside T and W

We recall that the wild kangaroo footsteps are in a region called W and the tame kangaroo footsteps are in an region called T, and they intersect in an region called A. Now we analyse the number of tame kangaroo footsteps that are outside the bounds of T and the number of wild kangaroo footsteps that are outside W. Note that T and W have the same area (i.e., the same area as $\mathcal{X}$), because both of them have sides $(-da, da)$, $(-cb, cb)$. The only difference is that T is centered in the centre of $\mathcal{X}$ while W is centered in Q. Therefore, the average number of wild kangaroo footprints outside W is the same as the average number of tame kangaroo footprints outside T.

Our analysis reduces to a 1-dimensional random walk. In the y-direction, the average length is θ. Since the steps are always increased by 1 and the most distant starting point from the centre of T (or W) is cb, for $c = 1 - \theta/b$ (see Lemma 5.5.1), we can assume that the steps do not go outside T (or W) in the y-direction. Now we analyse the x-direction.

Conjecture 4. *Let T (or W) be the region where the tame (respectively wild) random walks are supposed to lie and let K be the total number of steps. We conjecture that the number of steps outside the bounds of T (respectively W) is $N_2 \leqslant 0.025K$.*

Evidence for the conjecture. Experimentally, we used MAGMA [10] to perform an experiment to support our conjecture. We fixed the starting point to be $a - \beta(Dist)$ for some positive integer β and computed the number of steps outside the bounds of T (respectively W). We recall from Lemma 5.5.5.1 that $Dist$ is the expected maximum distance from the starting point after θ steps. We repeated the experiment 1000 times for each β in $[1, a/Dist]$. The results are shown in Table 5.4. We call P_β the probability of steps being outside $\mathcal{X}$ for each β (we measured that by counting the number of points $[x]P_1 + [y]P_2$ in which $|x| > a$ and dividing the result by the total number of steps).

Table 5.4: Number of steps outside T (or W)/ Total number of steps.

	$t = 26, \theta = 2^5$	$t = 24, \theta = 2^7$	$t = 22, \theta = 2^8$	$t = 26, \theta = 2^{15}$
β	P_β	P_β	P_β	P_β
1	0.29	0.26	0.26	0.29
2	0.19	0.16	0.15	0.14
3	0.10	0.11	0.08	0.08
4	0.05	0.05	0.03	0.06
5	0.04	0.03	0.035	0.02
6	0.03	0.008	0.002	0.01
7	0.02	0.0004	0.002	0.003
8	0.01	0.01	0.001	0.001
9	0.005	0.01	0.0009	0.003
10	0.004	0.02	0	0.009
11	0.004	0.006	0	0.008
12	0.0002	0.005	0	0.001
13	0.0002	0	0	0
14	0	0	0	0
15	0	0	0	0
P_{out}	0.0215	0.021	0.013	0.018

From Table 5.4, we computed the overall probability of steps outside T (respectively W). For $\beta \geqslant 14$ we always found $P_\beta = 0$. The values used arise from Section 5.6. For the second, third and forth columns we took $L = 30$ and $a = 98304$. For the fifth column we took $L = 40$ and $a = 3145728$. The value of $Dist$ depends on θ and t, since $Dist \approx m\sqrt{2\theta/\pi}$.

The probability that steps are outside T (or W) is

$$P_{out} = \frac{Dist}{a} \sum_{\beta=1}^{\lfloor a/Dist \rfloor} P_\beta.$$

The results of the last row support our conjecture. Since the total number of steps is K, it follows that

$$N_2 = KP_{out} \leqslant 0.025K.$$

We performed a second experiment in that we started the walks from a random starting point in $(-da, da) \times (-cb, cb)$ and computed the number of steps outside the bounds. We used $\theta = 2^5$ and $a = 98304$ and the maximum value found in 1000 repetitions was $0.02K$, which confirms the first experiment.

From these experiments, we concluded that only walks starting very near the bounds can cause some problems to our analysis.

5.5.7 Choice of Q and θ

We recall from Section 5.5.4 that some walks may not reach a distinguished point and also some steps can be outside the bounds of T and W. Then in Section 5.5.6 we estimated the values N_1 and N_2, which respectively represents the number of steps abandoned when a walk does not reach a distinguished point and the number of steps of the tame or wild kangaroos that are outside the bounds of T or W.

Let $\alpha \in \mathbb{R}$ be a constant such that $\#A = \alpha(\#\mathcal{X})$. Gaudry and Schost [25] give an estimate of A in relation to $\mathcal{X}$. Clearly, $A \neq \emptyset$. In the worst case, Q is in one of the corners of $\mathcal{X}$, and $\#A = \frac{1}{4}(\#\mathcal{X})$. In the best case, Q is close to the centre of $\mathcal{X}$, and $\#A \approx \#\mathcal{X}$. So, we take $\#A = \alpha(\#\mathcal{X})$ for some α in $[1/4, 1]$.

Note that if we have k' steps in A, assuming the distribution uniform, we have approximately $k'/2$ tame footsteps and $k'/2$ wild footsteps in A. Also note that the total number of good steps (i.e., k) is the sum of the good steps in T and the good steps in W. Since T and W have the same area as $\mathcal{X}$, and $\#A = \alpha(\#\mathcal{X})$, we have $k'/(2\alpha)$ good steps in T and $k'/(2\alpha)$ good steps in W. Therefore $k = k'/\alpha$.

We computed in Section 5.5.5.4 the probability of at least one successful Tame-Wild collision in an area A, which is the area in that the tame kangaroos steps intersect the wild kangaroos steps.

Lemma 5.5.6. *Let notation be as above, so $\alpha = \frac{\#A}{\#\mathcal{X}}$. The required number of good steps is $k = \frac{1}{\alpha}\sqrt{(\#\mathcal{X})\ln 2}$.*

Proof. We cover $\mathcal{X}$ close to uniformly, with k kangaroo footprints. Then the number of steps which lie in A is approximately αk. To have $\alpha k = \sqrt{(\#\mathcal{X})\ln 2}$, we need $k = \frac{1}{\alpha}\sqrt{(\#\mathcal{X})\ln 2}$. □

Finally, we can state the following theorem:

Theorem 5.5.7. *Let $\mathcal{X}$ be the sample space and let $\#\mathcal{X}$ be the number of elements in $\mathcal{X}$. Assume Conjecture 4 above. The total number of steps of Algorithm 6 to have success probability > 0.5 in the worst case is at most $3.51\sqrt{\#\mathcal{X}}$ group operations.*

Proof. We know from Section 5.5.5.3 that the total number of steps of Algorithm 6 is $\mathbf{Q}\theta$. From Lemma 5.5.6, we recall that

$$k = \frac{1}{\alpha}\sqrt{(\#\mathcal{X})\ln 2}$$

is the total number of good steps.

To compute the total number of steps in Algorithm 6, we need to add N_1, and $2(N_2)$ to k (remember, the number of tame steps outside T is approximately the number of wild steps outside W). From Lemma 5.5.5, we have:

$$N_1 < (5 \times 10^{-8})K.$$

From Conjecture 4, we have:

$$N_2 \leqslant 0.025K.$$

So:

$$\begin{aligned} \mathbf{Q}\theta = K &\leqslant k + (5 \times 10^{-8})K + 2(0.025)K \\ K(1 - 5 \times 10^{-8} - 2(0.025)) &\leqslant k \\ K &\leqslant 1.05k \end{aligned}$$

We consider the worst case (i.e., $\alpha = 1/4$). Then:

$$K \leqslant 3.51\sqrt{\#\mathcal{X}}.$$

□

Now that we know the number of steps needed to find a useful collision (K), given the amount of memory available, one can compute the number of distinguished points necessary, i.e., $\mathbf{Q}$. In the parallel case, we let each processor compute $\mathbf{Q}/\mathbf{P}$ distinguished points.

Note that since we don't know where the discrete log is, we have to assume the worst case, that is, Q is in the corner of $\mathcal{X}$ and therefore, $A = \frac{1}{4}\mathcal{X}$. On average, the discrete log is more concentrated in the centre of $\mathcal{X}$ (remember the pictures of τ-adic distribution), so usually we take more steps than necessary. For example, in the best case, the discrete log is close to the centre of $\mathcal{X}$ (i.e., $\alpha \approx 1$ and $W = T = \mathcal{X}$) and if we take $K = 3.51\sqrt{\#\mathcal{X}}$

steps we have $k = 3.51/1.05\sqrt{\#\mathcal{X}}$ and hence $k' = \alpha k = 3.34\sqrt{\#\mathcal{X}}$. Using this value for k' in Equation (5.2) results in a success probability ≈ 0.94. The experimental results (see Section 5.6.1) confirm this value.

If we have any hint that $\alpha > 1/4$, we can take less steps and get a smaller constant hidden in the $O(\ )$.

5.6 Using the Gaudry-Schost Algorithm to Solve the τ-DLP

Now that we have described and analysed the Gaudry-Schost Algorithm, it is straightforward to use it to solve the τ-DLP. We take $P_1 = P$ for some point $P \in E(\mathbb{F}_{2^m})$ and $P_2 = \tau(P)$. We know from Section 4.2.4 that a τ-adic expansion of length L can be mapped down to $x + y\tau$, for integers x, y of certain size. We recall from Section 5.5.7 that the total number of steps of Algorithm 6 depends on the number of elements in $\mathcal{X}$.

Using the bounds for x and y (recall Theorem 4.2.2: $|x| < a = 2.81\sqrt{2^L}$ and $|y| < b = 2\sqrt{2^L}$) leads to $\#\mathcal{X} = 4ab = 22.48(2^L)$. Therefore, from Theorem 5.5.7, Section 5.5.7, the required number of group operations when $\alpha = 1/4$ is $2^{L/2+4.06}$.

Using the τ-adic distribution (see Conjecture 1, Section 4.2.6), for the general τ-DLP we take $a = 2.28\sqrt{2^L}$ and $b = 1.8\sqrt{2^L}$, which leads to $\#\mathcal{X} = 4ab = 16.42(2^L)$. From Theorem 5.5.7, Section 5.5.7, the required number of group operations when $\alpha = 1/4$ is $2^{L/2+3.83}$.

We can use the same algorithm for the general τ-DLP to solve the τ-NAF

DLP. Obviously, the output will not be a τ-NAF, but will be an expression $x + y\tau$ equivalent to the τ-NAF which represents the discrete log. Using the bounds proved for τ-NAFs (recall Theorem 4.2.3: $|x| < a = 2.20\sqrt{2^L}$ and $|y| < b = 1.52\sqrt{2^L}$) results in $\#\mathcal{X} = 4(2.20)(1.52)2^L = 13.38(2^L)$). In that case, the number of group operations according to Theorem 5.5.7 when $\alpha = 1/4$ is $2^{L/2+3.68}$.

We can get an even smaller constant implicit in the $O(\ )$ when we make use of the τ-NAF distribution (remember, according to Conjecture 2, for τ-NAFs we have $|x| < a = 1.53\sqrt{2^L}$ and $|y| < b = 1.24\sqrt{2^L}$, which leads to $\#\mathcal{X} = 4(1.53)(1.24)2^L = 7.59(2^L)$). As a consequence, we can choose a smaller number of processors and the number of group operations becomes $2^{L/2+3.27}$ when $\alpha = 1/4$.

5.6.1 Experimental Results

We implemented Algorithm 6 (which we call $\mathcal{A}_\tau$ for short) in MAGMA [10] and obtained the following results (see Tables 5.5 and 5.6):

Experiment 1 Run $\mathcal{A}_\tau$ N times, each time picking a random τ-adic n of length $\leqslant L$ such that $Q = [n]P$.

Obs. (1) We used the values of a and b based on the distribution of τ-adics (see Figures 4.2 and 4.3);
Obs. (2) We used the values of a and b based on the distribution of τ-NAFs (see Figure 4.6).

Note that the success probability gets smaller when we use the τ-adic distribution. This can be explained because when we choose $a = 3\sqrt{2^L}$ and $b = 2\sqrt{2^L}$ (such values are an approximation to integers of the

Table 5.5: Experimental results - type 1.

L	N	a	b	$\mathbf{Q}\theta$	Average run time	# of success	Obs
30	100	$3\sqrt{2^L}$	$2\sqrt{2^L}$	$2^{19.10}$	$2^{18.56}$	96	
30	100	$3\sqrt{2^L}$	$2\sqrt{2^L}$	$2^{19.10}$	$2^{18.65}$	88	
30	100	$3\sqrt{2^L}$	$2\sqrt{2^L}$	$2^{19.10}$	$2^{18.57}$	94	
30	100	$2.28\sqrt{2^L}$	$1.8\sqrt{2^L}$	$2^{18.82}$	$2^{18.43}$	84	(1)
30	100	$2.28\sqrt{2^L}$	$1.8\sqrt{2^L}$	$2^{18.82}$	$2^{18.44}$	86	(1)
30	100	$2.28\sqrt{2^L}$	$1.8\sqrt{2^L}$	$2^{18.82}$	$2^{18.50}$	77	(1)
40	100	$2.28\sqrt{2^L}$	$1.8\sqrt{2^L}$	$2^{23.82}$	$2^{23.42}$	80	(1)
30	100	$1.53\sqrt{2^L}$	$1.24\sqrt{2^L}$	$2^{18.27}$	$2^{18.06}$	62	(2)

results of Theorem 4.2.2) we do not expect to have the DLP very close to the edges and therefore, the overlap region $\mathcal{A}$ will be large with high probability. However, when we reduce the region based on the τ-adic distribution, it can happen that the DLP is in the edges, so the worst case (i.e., the discrete log close to one of the corners of $\mathcal{X}$ and hence a smaller $\mathcal{A}$) is more likely to occur.

Experiment 2 Pick a random τ-adic n of length $\leqslant L$ such that $Q = [n]P$ and afterwards run $\mathcal{A}_\tau$ N times, for the same Q.

Table 5.6: Experimental results - type 2.

L	N	a	b	$\mathbf{Q}\theta$	Average run time	# of success	Obs
30	100	$3\sqrt{2^L}$	$2\sqrt{2^L}$	$2^{19.10}$	$2^{18.58}$	94	
30	100	$3\sqrt{2^L}$	$2\sqrt{2^L}$	$2^{19.10}$	$2^{18.57}$	96	
30	100	$2.28\sqrt{2^L}$	$1.8\sqrt{2^L}$	$2^{18.82}$	$2^{18.51}$	72	(1)
30	100	$2.28\sqrt{2^L}$	$1.8\sqrt{2^L}$	$2^{18.82}$	$2^{18.42}$	86	(1)
30	100	$1.53\sqrt{2^L}$	$1.24\sqrt{2^L}$	$2^{18.27}$	$2^{18.11}$	61	(2)

The same Obs. of Table 5.5 hold for Table 5.6.

Experiment 3 We fixed Q in one of the corners of $\mathcal{X}$ (i.e., we took $Q = a + b\tau$) and ran $\mathcal{A}_\tau$ 100 times. We got 59 successes, which is close to the expected theoretical probability of success. We recall that we had assumed that Q is in one of the corners to compute the theoretical probability of success. In practice, however, the discrete log is more likely to be close to the centre of $\mathcal{X}$ (see distribution of τ-adics, Section 4.2.6), which explains why we got high probabilities with random choices, because we took more steps than necessary.

5.6.2 Further Remarks

Remark 5: If memory is available, then for some problems (e.g., τ-NAF DLP) it is better to do BSGS using τ-adics than to use the $x + y\tau$ representation, because the hidden constants are smaller.

Remark 6: Comparing the running time of Gaudry-Schost algorithm (Section 5.6) for the general τ-DLP with the running time of van Oorschot and Wiener algorithm for the weight-w τ-DLP (Table 5.3, Section 5.4) we conclude that depending on the value of w, it will be better to use the Gaudry-Schost 2-dimensional algorithm for the general τ-DLP to solve also the weight-w τ-DLP. For example, for $L = 140$ and $w = 24$, the required number of steps for solving the τ-DLP using van Oorschot and Wiener algorithm is approximately $2^{77.4}$ group operations and the required number of steps using 2-dim Gaudry-Schost algorithm with the same parameters is approximately $2^{74.06}$ group operations, when using the proved values $a = 2.81\sqrt{2^L}$ and $b = 2\sqrt{2^L}$ and even smaller when using sizes of a and b based on the τ-adic distribution. Remember, using Pollard-rho in the whole group $E(\mathbb{F}_{2^{163}})$ takes $2^{77.6}$ group operations, hence in this case, the Gaudry-Schost for the general τ-DLP is the best choice.

Remark 7: We do not believe that the Gaudry-Schost algorithm can take into account extra knowledge about the discrete logarithm (i.e., that it has Hamming weight w). This is analogous to the fact that there is no low memory algorithm better than [57] for the standard low Hamming weight DLP.

Remark 8: The improvement to Pollard methods using equivalence classes under Frobenius (see [24, 60]) does not seem to apply in this case since our small interval typically contains only one solution within a Frobenius orbit of Q.

5.7 Summary

In this chapter we defined three variants of the Frobenius expansion DLP, namely, the general τ-DLP, the τ-NAF DLP and the weight-w τ-DLP and presented algorithms to solve them. We are particularly interested in cases when the length L of the Frobenius expansion is much smaller than the field size, which means that solving the standard DLP and then, if necessary, computing the Frobenius expansion, is not considered, since the standard DLP is believed to be hard.

In particular, we used the Gaudry-Schost algorithm to solve the general τ-DLP and the τ-NAF DLP in $2^{L/2+\varepsilon}$ group operations with a theoretical success probability around 0.54. The value of ε varies according to the position of the discrete log and can be improved if we consider the distributions of τ-adics and τ-NAFs. We experimented in practice using small parameters and achieved a success probability bigger than 0.8.

Such difference between theory and practice can be explained because in the computation of the theoretical probability of success, we have to assume the worst case, that is, Q is in the corner of $\mathcal{X}$ and therefore, the overlap region is minimal. On average, the discrete log is more concentrated in the centre of $\mathcal{X}$, so for random choices we usually take more steps than necessary and hence the success probability in practice is bigger than the probability expected in theory. If we have any hint that $\alpha > 1/4$, we can take less steps and get a smaller constant hidden in the $O(\ )$. We experimented fixing Q in one of the corners of $\mathcal{X}$ and got a success probability around 0.59, as expected.

Chapter 6

The GPS Identification Scheme Using Frobenius Expansions

Contents

In this chapter we present an application of Frobenius expansions with the GPS identification scheme. We begin recalling the original scheme, proposed by M. Girault and proved secure by G. Poupard and J. Stern. We also remark that a natural implementation is to use the scheme over a Koblitz curve and convert integers into Frobenius expansions to perform faster scalar multiplication. Then, we present τ-GPS, using an idea proposed by H. Lenstra, in that scalars are replaced by Frobenius expansions in protocols. We also present a security analysis of τ-GPS and describe a statistical attack over the τ-GPS protocol if the randomisation step proposed in Section 4.3 is not used.

6.1 Motivation

The GPS[1] identification protocol was the starting point of our entire project. At the beginning, we were tempted to use Frobenius expansions with short exponents in the GPS protocol, which would lead to smaller parameters, compared with the usual integer case, because we believed, at that point, that low-memory counterparts for the τ-DLP did not exist, or at least, did not have a square-root behaviour.

We were proven wrong, as we showed in Chapter 5, but the τ-GPS protocol may still have some advantages. One can speed up the offline operations on Koblitz curves significantly by using Frobenius expansions to compute the required point multiplications. However, to implement the GPS protocol would require conversion between Frobenius expansions and integers and this would lead to extra code on the device (i.e., silicon area) and extra computational cost.

Hence, the motivation of the present chapter is to develop a GPS system which uses Frobenius expansions throughout. This will lead to fast and

[1]The abbreviation “GPS” stands for its authors names, **G**irault, **P**oupard and **S**tern, so there is no connections at all with GPS, the Global Positioning System.

simple offline operations while still keeping the online operation fast (though the arithmetic of the online operation is more complicated than standard integer arithmetic and so is not as fast as standard GPS).

Since the study of τ-DLP showed that we cannot use shorter parameters, the main advantage of using Frobenius expansions is that it does not require conversion between integers and Frobenius expansions, being useful to applications with limited offline computation time and limited code area.

6.2 The GPS Identification Scheme

The GPS public key identification scheme is a three move challenge-response protocol based on the Schnorr signature scheme (see Section 2.9 and [46]). It was first described by Girault [26] and later developed by Poupard and Stern [43] (also see [28]).

Recall that a public key identification scheme allows a prover to convince a verifier that she possess the private key. In a three move protocol, the prover sends a *commitment* (which can be computed in advance, i.e. offline), then receives a *challenge* and answers with a *response* (this is the "online step", which must be performed in realtime). The verifier then performs a computation involving the commitment, challenge, response and public key and outputs either *accept* or *reject*.

The idea of the GPS scheme is to make the online phase as fast and simple as possible, so that it can be easily performed by very low power devices. Further improvements to speed up the online phase were proposed by Girault and Lefranc [27]. Okamoto, Katsuno and Okamoto [40] give an approach to reduce the bandwidth of the online step at the expense of the size of the public key.

A typical application of GPS (see [28]) is "on-the-fly" authentication at a road toll. Each car has a low cost smart card in which a GPS protocol runs. The time required to transmit data and to perform online calculations is very small, hence the car does not need to stop at the toll to be authenticated. The offline operations may be calculated while the car is driving along. Note that we assume that the verifier (i.e., toll gate) has significant computational resources, so that the verification step can be done quickly.

The original proposals for the GPS scheme suggested working in the group $(\mathbb{Z}/N\mathbb{Z})^*$ where N is an RSA modulus, or $\mathbb{F}_p^*$ where p is a large prime. However, since the computational device has extremely limited power it is more natural to work with elliptic curves, especially Koblitz curves over finite fields of small characteristic. The GPS protocol as described in [28] can be implemented with such elliptic curves, and the security results apply to this case. Using elliptic curves can give a significant speed-up to the offline generation of the commitment and the verification step, as well as having lower memory and power consumption requirements. Due to the nature of the GPS protocol, using elliptic curves does not have any effect on the running time of the online step.

6.3 The Original GPS Scheme

We recall from Section 2.9 that a modular reduction is needed in the Schnorr scheme. The main idea of the GPS protocol is to eliminate the modular reduction performed during the response step. In other words, the response is just $y = r + sc$. This makes the computation more efficient, and it reduces the code footprint on the device (since there is no need to implement arithmetic modulo $\mathbf{r}$).

The modular reduction in the computation of y is used to prove the zero knowledge property of Schnorr signatures, hence a new proof of security is required for GPS and the parameters have to be carefully chosen (see [28, 43]). In [28] the interactive protocol is proven to have statistical zero knowledge. The original proposal by Girault [26] used the group $(\mathbb{Z}/N\mathbb{Z})^*$ where N is an RSA modulus. Later work [28] proposed any cyclic group for which the discrete logarithm problem is hard.

An improvement to the original scheme, for which the online step is just a single addition, was presented by Girault and Lefranc [27]. It requires that the challenge (or alternatively, the private key) have a specific sparse form. Table 6.1 gives the reader an idea of the bitlengths of the integers (s, c, r) for an 80-bit security level of the private key and probability of successful forgery at most $1/2^{35}$. For more details see [27, 28].

Table 6.1: Numerical example of GPS scheme.

Scheme	bitlength of s	bitlength of c	bitlength of y
Standard GPS	160	35	275
Girault-Lefranc	160	940	1180

6.4 GPS on Koblitz Curves With Fast Scalar Multiplication

The GPS scheme can be implemented using elliptic curves over a finite field. In particular, one can use a Koblitz curve and convert integers into Frobe-

nius expansions to perform fast scalar multiplications. We briefly give the details:
Take a public point P and generate the key pair $\{s, I = [-s]P\}$. For simplicity of notation, let us denote a τ-adic representation of an integer x by $\tilde{x}$. The prover picks a random integer r, converts it into τ-adic $\tilde{r}$, computes the commitment $X = [\tilde{r}]P$ and sends X to verifier. Then, the verifier picks a random integer c and returns it to prover. The answer step is the same as the standard GPS (i.e., compute $y = r + sc \in \mathbb{Z}$) and the verification becomes to check if $X = [y]P + [c]I$. Again, to increase performance, the verification step can be computed as $[\tilde{y}]P + [\tilde{c}]I$.

This method is very efficient, however there is the additional cost of converting an integer to a Frobenius expansion (plus the extra code footprint this requires). As noted by Solinas [50, 51], in any cryptographic protocol, instead of choosing a random integer and converting to a τ-adic expansion one can directly choose a τ-adic. This idea could be used for r in the standard GPS protocol, but one would still need to convert back to an integer for the computation $y = r + sc \in \mathbb{Z}$ in the online step. As mentioned earlier, the conversion algorithm requires modular arithmetic which is not otherwise needed as part of the GPS protocol. This results in additional overhead in running time and code on the device. We will see in the next section that these additional costs can be avoided if we use random τ-adics instead of random integers.

6.5 τ-GPS

When using τ-adic expansions instead of integers, we will get much faster computations for $I = [-s]P$, $X = [r]P$ and, in the verification step, for $X = [y]P + [c]I$. Figure 6.1 shows the τ-GPS scheme. We recall from Definition 17 (Section 4.2) that $\mathcal{T}_n = \left\{ \sum_{i=0}^{n-1} x_i \tau^i \mid x_j \in \{-1, 0, 1\} \right\}$. We can repeat the protocol l times (though usually $l = 1$). We represent an element x picked at random from a set $\mathcal{X}$ by: $x \overset{\mathsf{r}}{\longleftarrow} \mathcal{X}$.

System Parameters:		$\mathcal{S}, \mathcal{R}, \mathcal{C} \in \mathbb{N}$, $P \in E(\mathbb{F}_{2^m})$
Private key:		$s \overset{\mathsf{r}}{\longleftarrow} \mathcal{T}_{\mathcal{S}}$
Public key:		$I = [-s]P$
PROVER		VERIFIER
Commitment		
$r \overset{\mathsf{r}}{\longleftarrow} \mathcal{T}_{\mathcal{R}}$		
$X = [r]P$	$\xrightarrow{X}$	
		Challenge
	$\xleftarrow{c}$	$c \overset{\mathsf{r}}{\longleftarrow} \mathcal{T}_{\mathcal{C}}$
Response		
if $c \notin \mathcal{T}_{\mathcal{C}}$ then **abort**		
compute $y = r + sc$		
if $y \notin \mathcal{T}_{\Phi,\mathcal{R}}$ then **abort**	$\xrightarrow{y}$	
		if $y \notin \mathcal{T}_{\Phi,\mathcal{R}}$ then **reject**
		if $X = [y]P + [c]I$ then **accept**
		else **reject**.

Figure 6.1: τ-GPS

As with the original GPS protocol it is essential for the prover to perform a size check on the challenge c (otherwise, a dishonest verifier can send, for example, $c = \tau^{\mathcal{R}}$ and recover the prover's secret). The size check on y does not seem to be essential for security, but we include it to ease the security analysis. Heuristics 1 and 2 (see Section 4.4) imply that the probability of aborting in the protocol is negligible.

Note that the computation of $s \times c$ can be performed using the non-randomised version of Algorithm 4 (see Section 4.3). We recall from Section 4.3.1 that $\mathcal{K}'$ is a security parameter. We will show in Section 6.6.3.1 that the extra randomisation of the $\mathcal{K}'$ lowest order coefficients is only required when performing the addition with r. For further implementation details see Chapter 7.

6.6 Security Analysis

We closely follow [28] for our security analysis. In particular, we use the same security model as in [28]. We first consider attacks which recover the private key (i.e., solve the discrete log). Then, our analysis follows the approach of [22] and proves *completeness*, *soundness* and *zero knowledge*.

6.6.1 Discrete Logarithms

Given (P, I) it must be infeasible for an attacker to compute the private key s (similarly, given (P, X) to compute r). One could solve the DLP to get $\lambda \in \mathbb{N}$ such that $-I = [\lambda]P$ and then convert λ to a Frobenius expansion. Hence, we require the DLP in $\langle P \rangle$ to be hard. As we are working with Koblitz curves (see Chapter 4), we can take, in practice, $m = 163$. It is easy to check that our curve over $\mathbb{F}_{2^{163}}$ has a nearly prime order, as needed.

Alternatively, one could try to solve the τ-DLP (see Section 5.2), i.e., compute $s \in \mathcal{T}_{\mathcal{S}}$ directly from (P, I). More precisely:

We assume that $\mathcal{PP}(\omega_{\mathcal{PP}}, k)$ is a randomised algorithm that generates public parameters P and $I = [-s]P$ for $s \in \mathcal{T}_{\mathcal{S}}$ under a random tape $\omega_{\mathcal{PP}}$ and with the security parameter k as an input. We recall the *τ-adic Discrete Logarithm Problem* as: given inputs P, Q and $\mathcal{S}$, to find $x \in \mathcal{T}_{\mathcal{S}}$ (if it exists) such that $Q = [x]P$.

Definition 30. *The* τ-DLP Assumption *is that for every polynomial $\mathcal{Q}_1$ and every probabilistic polynomial time (PPT) Turing Machine $\mathcal{A}$ running on random tape $\omega_{\mathcal{A}}$, for sufficiently large k,*

$$\boldsymbol{Pr}\big[\mathcal{A}(P, \mathcal{S}, Q) = x\big] < \frac{1}{\mathcal{Q}_1(k)}$$

where $(P, \mathcal{S}, Q) \leftarrow \mathcal{PP}(\omega_{\mathcal{PP}}, k)$, $Q = [x]P$ and $x \in \mathcal{T}_{\mathcal{S}}$.

6.6.2 Completeness

We want to show that GPS with τ-adic expansions is complete. We recall from Section 4.4 that $\Phi = \mathcal{S} + \mathcal{C} + 3$.

Theorem 6.6.1 (Completeness)**.** *Suppose $\mathcal{R} \geqslant \Phi + \mathcal{K}$, for sufficiently large $\mathcal{K} \in \mathbb{N}$. Then, a prover who possess a valid key pair $(s, I = [-s]P)$ is accepted with overwhelming probability by a verifier.*

Proof. Clearly, the prover can compute $y = r + sc$. The verification step is:

$$[y]P + [c]I = [r + sc]P + [c][-s]P = [r]P + [sc]P + [-sc]P = [r]P = X,$$

which is successful. Finally, since $\mathcal{R} \geqslant \Phi + \mathcal{K}$, from Heuristic 1 (Section 4.4) we expect that $y \in \mathcal{T}_{\mathcal{R}}$ with overwhelming probability. $\square$

6.6.3 The Zero Knowledge Proof

As with any public key identification or signature scheme it is important to show that runs of the protocol do not leak information about the private key. For the original GPS scheme it is proved in [28] that the protocol has statistical zero knowledge.

The security of the τ-GPS scheme is more subtle than the integer case. First we present an attack on the basic scheme. The attack does not have an analogue in the standard GPS setting. This motivates the extra randomisation in the computation of $y = r + sc$.

We have not been able to prove statistical zero knowledge of our scheme; instead, we prove computational zero knowledge with respect to a computational assumption.

6.6.3.1 Obtaining Information About the Private Key

Before proving zero knowledge, we remark that the τ-GPS protocol does leak information about s if one omits the extra randomisation step in the computation $y = r + sc$ (see Section 4.3.1).

We give a brief sketch of the idea. Suppose a dishonest verifier inspects the constant coefficients of all polynomials, and always chooses c such that $c_0 = 1$. Then $y_0 = r_0 + s_0$, where $+$ is addition using Algorithm 4, so y_0 is reduced to the set $\{-1, 0, 1\}$. If r_0 is uniformly distributed then the distribution of y_0 depends on the value of s_0. Table 6.2 shows the possible outcomes for y_0 if the optional randomisation step is not used and the adversary always chooses $c_0 = 1$ (similar tables can be constructed if the adversary always chooses $c_0 = -1$).

Table 6.2: Statistical attack to recover s_0.

s_0	r_0	$y_0 = s_0 + r_0$
-1	-1	0
	0	-1
	1	0
0	-1	-1
	0	0
	1	1
1	-1	0
	0	1
	1	0

More precisely, if $s_0 = -1$ then the output distribution of y_0 in this case ($c_0 = 1$) is 0 with probability 2/3 and -1 with probability 1/3; if $s_0 = 0$ then y_0 has uniform distribution in $\{-1, 0, 1\}$; if $s_0 = 1$ then $y_0 = 0$ with probability 2/3 and 1 with probability 1/3. After a sufficient number of repetitions, an adversary finds the coefficient s_0 of the private key by observing the distribution of y_0.

We now explain how the attack can be continued to obtain more information about the private key. Assume s_0 is known. Again, the adversary chooses $c_0 = c_1 = 1$. From $y = sc + r$, we have $y_1 = s_0 + s_1 + r_1 + \varepsilon$, where ε is the carry produced from computing $s_0 + r_0$. We recall that y_0 is reduced to the set $\{-1, 0, 1\}$, so if $r_0 = s_0 = 1$, then $y_0 = 2$ and we have to add $-\tau^2 + t\tau - 2$ (remember, $\tau^2 - t\tau + 2 \equiv 0$), which produces a carry $\varepsilon = t$; if $r_0 = s_0 = -1$, then $y_0 = -2$ and $\varepsilon = -t$.

We stress that the addition is computed using Algorithm 4 and therefore, it is not associative (e.g., $(1 + 1) + (-1) = 0 + (-1) = -1$ and $1 + (1 + (-1)) = 1 + 0 = 1$). Hence, we assume that one first computes $(s_0 + s_1)$

and then computes $r_1 + (s_0 + s_1)$ and finally, if there is a carry ε, computes $(r_1 + (s_0 + s_1)) + \varepsilon$.

The attack proceeds in the same way the adversary did to recover s_0, i.e., he runs the algorithm many times, always choosing a challenge c such that $c_0 = c_1 = 1$ and then he constructs a table with all values of y_1. After a sufficient number of repetitions, the adversary analyses the distribution of y_1. If necessary, he chooses other values for c_0 and c_1 and repeats the attack, until he is be able to recover s_1.

Note that in the attack to recover s_0, the adversary does not recover the value of r_0. If from the first attack, the adversary knows that $s_0 = 0$, then clearly there is no carry to the next coefficient. We now analyse that case: the adversary knows $s_0 = 0$, then he chooses $c_0 = c_1 = 1$ and after a number of repetitions, he analyses the distribution of y_1. Table 6.3 shows the distribution of y_1 when $s_0 = 0$, $c_0 = c_1 = 1$.

Table 6.3: Statistical attack to recover s_1 - in this case, the adversary knows that $s_0 = 0$ and chooses $c_0 = c_1 = 1$.

s_0	s_1	r_1	$y_1 = (s_0 + s_1) + r_1$
0	-1	-1	0
		0	-1
		1	0
	0	-1	-1
		0	0
		1	1
	1	-1	0
		0	1
		1	0

From Table 6.3, if the output distribution of y_1 is 0 with probability 2/3 and -1 with probability 1/3 then $s_1 = -1$; if y_1 has distribution close to uniform then $s_1 = 0$; if y_1 is 0 with probability 2/3 and 1 with probability

1/3 then $s_1 = 1$.

Now we consider the cases when it is possible that a carry $\varepsilon \neq 0$ occurs in the computation of y_1, i.e., the cases when $s_0 = -1$ or $s_0 = 1$. We begin with $s_0 = -1$. The adversary knows that when $s_0 = -1$, if $r_0 = 1$ or $r_0 = -1$, the output y_0 (remember $y_0 = s_0 + r_0$) will be 0. So, he learns nothing about r_0. However, he knows that when $y_0 = -1$ (which happens with probability 1/3 from Table 6.2) then $r_0 = 0$ and there will be no carry to y_1. Hence, the adversary throws away all cases with $y_0 = 0$ and only analyses the outputs y_1 when $y_0 = -1$. The distribution of such y_1 is in Table 6.4.

Table 6.4: Statistical attack to recover s_1 - in this case, the adversary knows that $s_0 = -1$ and analyses only the cases when $y_0 = -1$. Again, he chooses $c_0 = c_1 = 1$.

s_0	s_1	r_1	$y_1 = (s_0 + s_1) + r_1$
-1	-1	-1	-1
		0	0
		1	1
	0	-1	0
		0	-1
		1	0
	1	-1	-1
		0	0
		1	1

From Table 6.4, if the output distribution of y_1 is 0 with probability 2/3 and -1 with probability 1/3 then $s_1 = 0$; if y_1 has distribution close to uniform then s_1 can be either 1 or -1 and the adversary needs to construct another table using different values for c_0 and/or c_1, as we now show.

Table 6.5 shows the new distribution of y_1 if the adversary, after observing that the distribution of y_1 is close to uniform in Table 6.4, chooses $c_0 = 1$ and $c_1 = 0$. The output distribution of y_1 will be 0 with probability 2/3 and -1 with probability 1/3 if $s_1 = -1$ and will be 0 with probability

2/3 and 1 with probability 1/3 if $s_1 = 1$. Hence, after a sufficient number of repetitions, the adversary will be able to determine the coefficient s_1.

Table 6.5: Statistical attack to recover s_1 – part 2.

s_1	r_1	$y_1 = s_1 + r_1$
-1	-1	0
	0	-1
	-1	0
1	-1	0
	0	1
	1	0

Now we analyse for $s_0 = 1$. Again, the adversary knows that when $s_0 = 1$, if $r_0 = 1$ or $r_0 = -1$, the output y_0 (remember $y_0 = s_0 + r_0$) will be 0. So, he learns nothing about r_0. However, he knows that when $y_0 = 1$ (which will occur with probability 1/3 from Table 6.2) then $r_0 = 0$ and there will be no carry to y_1. Hence, the adversary analyses only the outputs y_1 for which $y_0 = 1$. The distribution of such y_1 is very similar to the distribution of y_1 in Table 6.4, so we do not give a table.

If the output distribution of y_1 is 0 with probability 2/3 and 1 with probability 1/3 then $s_1 = 0$; if y_1 has distribution close to uniform then s_1 can be either 1 or -1 and the adversary needs to construct another table (very similar to Table 6.5) using different values for c_0 and/or c_1.

The above method can be extended to recover the first few coefficients of s. However, the randomisation of the first $\mathcal{K}'$ coefficients destroys the attack if $\mathcal{K}'$ is sufficiently large. Similarly, it seems hard to mount the attack on the $\mathcal{K}'$-th coefficient, since that is influenced by carry values propagating from addition of lower-degree terms (for example, $y_2 = r_2 + s_2c_0 + s_0c_2 + s_1c_1 + \varepsilon_1 + \varepsilon_2$, where ε_1 is the possible carry that comes from $s_0 + r_0$ and ε_2 is the possible carry that comes from $((s_1 + s_0) + r_1) + \varepsilon_1)$.

6.6.3.2 Trying to Prove Statistical Zero Knowledge

Our first attempt was to prove the statistical zero knowledge property of our scheme, depending on the heuristics listed earlier, but due to the non canonical arithmetic of Frobenius expansions, we were unable to obtain such a proof. In this subsection we explain what went wrong.

Let $\mathcal{K}$ and $\mathcal{K}'$ be security parameters and define $\mathcal{R} = \max\{\Phi+\mathcal{K}, m+\mathcal{K}'\}$. Assume Heuristics 1 and 2 from Section 4.4. We tried to prove that the GPS protocol is statistically zero-knowledge if l and $3^{\mathcal{C}}$ are polynomial and if $\mathcal{K}$ and $\mathcal{K}'$ are sufficiently large.

We closely followed the proof of Theorem 2 of [28], but we could not use exactly the same proof due to the strange properties of addition of Frobenius expansions. Let $\mathcal{A}_1$ be a dishonest verifier, who instead of picking challenges at random, chooses them based on previous iterations with a legitimate prover, in order to try to obtain some knowledge about the private key. Following [28] we denote by $c(X, hist, \omega_{\mathcal{A}})$ the challenge chosen, depending on the commitment X, the history $hist$ of the protocol so far, and the random tape $\omega_{\mathcal{A}}$.

We now define an algorithm which simulates a round, using a random tape $\omega_{\mathcal{A}}$. We recall from Section 4.4 that $\Phi = \mathcal{S} + \mathcal{C} + 3$ and the definition of $\mathcal{T}_{\Phi,\mathcal{R}}$. Note that $\#\mathcal{T}_{\Phi,\mathcal{R}} = 3^{\mathcal{R}} - 3^{\Phi}$. The simulation is:

Step 1. Use $\omega_{\mathcal{A}}$ to choose random values $\bar{c} \in \mathcal{T}_{\mathcal{C}}$ and $\bar{y} \in \mathcal{T}_{\Phi,\mathcal{R}}$.

Step 2. Compute $\bar{X} = [\bar{y}]P + [\bar{c}]I$

Step 3. If $c(\bar{X}, hist, \omega_{\mathcal{A}}) \neq \bar{c}$ then return to Step 1, else return $(\bar{X}, \bar{c}, \bar{y})$.

We should prove that for any fixed random tape $\omega_{\mathcal{A}}$, the triples $(\bar{X}, \bar{c}, \bar{y})$ output by the simulation are statistically indistinguishable from the real triples (X, c, y), even when c is controlled by a dishonest verifier.

When we say that triple $(\bar{X}, \bar{c}, \bar{y})$ is *statistically indistinguishable* from the real triple (X, c, y), it means that no algorithm which given a polynomial number of triples of both distributions can distinguish, with non-negligible advantage, which of the two distributions a triple belongs to, even when unlimited computational power is available.

Let $\mathcal{M} = E(\mathbb{F}_{2^m}) \times \mathcal{T}_{\mathcal{C}} \times \mathcal{T}_{\mathcal{R}}$. Fix a private key $s \in \mathcal{T}_{\mathcal{S}}$ and a public key $I = [-s]P$. Formally, we want to prove that

$$\Sigma_1 = \sum_{(\alpha,\beta,\gamma)\in\mathcal{M}} \left| \mathbf{Pr}_{\omega_{\mathcal{P}}}[(X, c, y) = (\alpha, \beta, \gamma) - \mathbf{Pr}_{\omega_{\mathcal{M}}}(\bar{X}, \bar{c}, \bar{y}) = (\alpha, \beta, \gamma)] \right|$$

is negligible.

First, we consider the simulation. Define $M_2 = \{(\alpha = \bar{y}P + \bar{c}I, \beta = \bar{c}, \gamma = \bar{y}) : \bar{c} \in \mathcal{T}_{\mathcal{C}}, \bar{y} \in \mathcal{T}_{\Phi,\mathcal{R}}$ and $\bar{c} = c(\bar{y}P + \bar{c}I, hist, \omega_{\mathcal{A}})\} \subset \mathcal{M}$. Note that the same triple (α, β, γ) cannot arise from more that one pair $(\bar{c}, \bar{y})$ (since the β and γ components fix $(\bar{c}, \bar{y})$ and hence the α components are equal. Clearly, the distribution on $\mathcal{M}$ arising from the simulation is the uniform distribution supported on $M_2 \subset \mathcal{M}$.

Now, we consider triples $(\alpha, \beta, \gamma) \in \mathcal{M}$ coming from genuine runs of the τ-GPS protocol. Define $M_1 = \{(\alpha = [r]P, \beta = c([r]P, hist, \omega_{\mathcal{A}}), \gamma = r + \beta s) : r \in \mathcal{T}_{\mathcal{R}}$ and $r + \beta s \in \mathcal{T}_{\mathcal{R}}\} \subset \mathcal{M}$. Note that the same value α above can arise from more than one value of $r \in \mathcal{T}_{\mathcal{R}}$. To continue with the proof, we expected that the exact same triple (α, β, γ) could not arise from two different values of r but this is not true in general as there can be equivalent but not equal r_1 and r_2 such that $r_1 + sc = r_2 + sc$ under the addition in Algorithm 4. Even without the randomisation this can happen, e.g., $r_1 = -1$, $r_2 = 1 - \tau + \tau^2$ and $sc = 1$ but the randomisation also gives further opportunities for it to happen.

Because of this, we need to know, for example, how often this can happen. The question is then for how many pairs (r_1, r_2) in the equivalence class of r can we have $r_1 + sc = r_2 + sc$ using Algorithm 4 with randomisation. Do solutions (r_1, r_2) exist for every equivalence class or only some of them? When it does happen, how many pairs of solutions can one get?

We performed an experiment using MAGMA[10] to count for each r_1, and fixed sc, how many $r_i \neq r_1$ are there such that $r_1 + sc = r_i + sc$. The results showed that the distribution of frequencies is quite irregular, varying, for example, from zero up to 14 distinct r_i for r_1 of length 10. Hence, it would be easy to distinguish uniformly chosen triples as used in the simulation from the real ones, with a simple frequency analysis, which spoiled our attempt to prove statistical zero knowledge.

We stress that these statistical irregularities do not seem to lead to an attack on the τ-GPS scheme.

We leave as an open problem to modify the τ-GPS scheme in order to make it statistical or even perfect zero knowledge.

6.6.3.3 Proving Computational Zero Knowledge

We now state the computational assumption on which our security result relies.

Definition 31. *Let $\mathcal{C}, \mathcal{S}, \mathcal{K}$ and $\mathcal{K}'$ be parameters. Define $\Phi = \mathcal{S} + \mathcal{C} + 3$ and $\mathcal{R} = \max\{\Phi + \mathcal{K}, m + \mathcal{K}'\}$. Let $s \in \mathcal{T}_{\mathcal{S}}$ and $I = [-s]P$. Let $\mathcal{M} = E(\mathbb{F}_{2^m}) \times \mathcal{T}_{\mathcal{C}} \times \mathcal{T}_{\mathcal{R}}$. Let $f : E(\mathbb{F}_{2^m}) \to \mathcal{T}_C$ be a function. Define the distribution on $\mathcal{M}$*

$$\mathcal{M}_{s,f,1} = \{(\alpha = [r]P, \beta = f(\alpha), \gamma = r + s\beta) : r \in \mathcal{T}_{\mathcal{R}}\}$$

where r is selected from $\mathcal{T}_{\mathcal{R}}$ uniformly at random and where the computation $r + s\beta$ is performed using Algorithm 4 with randomisation of the first $\mathcal{K}'$ coefficients. Define the distribution on $\mathcal{M}$

$$\mathcal{M}_{I,f,2} = \{(\alpha, \beta, \gamma) : \gamma \in \mathcal{T}_{\Phi,\mathcal{R}}, \alpha = [\gamma]P + [\beta]I, \beta = f(\alpha)\}$$

where γ is selected uniformly at random.

We stress that these distributions are not the same. For example, in $\mathcal{M}_{I,f,2}$ there can be points $\alpha \in E(\mathbb{F}_{2^m})$ which are not of the form $[r]P$ for some $r \in \mathcal{T}_{\mathcal{R}}$ (though they will typically be of the form $[r']P$ for some $r' \in \mathcal{T}_{\mathcal{R}'}$ where $\mathcal{R}' - \mathcal{R}$ is small). More importantly, the distributions on γ are not the same in both cases: in the latter case γ is uniform in $\mathcal{T}_{\Phi,\mathcal{R}}$ whereas due to the properties of addition using Algorithm 4 it is not clear whether the distribution of γ in the former case is close to uniform.

We now make a computational assumption regarding these distributions.

Assumption 1. *Let $\mathcal{A}$ be an algorithm running in polynomial time which is* *given I (but not s) and which samples elements from the distributions $\mathcal{M}_{s,f,1}$ and $\mathcal{M}_{I,f,2}$. Then we claim that $\mathcal{A}$ cannot distinguish the two distributions, namely that it does not have non-negligible advantage in being able to identify whether a given triple (α, β, γ) was drawn from $\mathcal{M}_{s,f,1}$ or $\mathcal{M}_{I,f,2}$.*

Theorem 6.6.2 (Zero Knowledge)**.** *Let $\mathcal{C}, \mathcal{S}, \mathcal{K}$ and $\mathcal{K}'$ be security parameters and define $\Phi = \mathcal{C} + \mathcal{S} + 3$ and $\mathcal{R} = \max\{\Phi + \mathcal{K}, m + \mathcal{K}'\}$. Assume Heuristics 1, 2 from Section 4.4 and Assumption 1 above. Then the τ-GPS protocol has computational zero-knowledge if l and $3^{\mathcal{C}}$ are polynomial and if $\mathcal{K}$ and $\mathcal{K}'$ are sufficiently large.*

Proof. We use exactly the same arguments used in Section 6.6.3.2, i.e., $\mathcal{A}_1$ is a dishonest verifier, who instead of picking challenges at random, chooses them based on previous iterations of the protocol, in order to try to obtain some knowledge about the private key. Following [28] we denote by $c(X, hist, \omega_{\mathcal{A}})$ the challenge chosen, depending on the commitment X, the history *hist* of the protocol so far, and the random tape $\omega_{\mathcal{A}}$.

The algorithm which simulates a round, using a random tape $\omega_{\mathcal{A}}$, without any knowledge of the secret s is also the same stated in Section 6.6.3.2 (see steps 1, 2 and 3). The protocol is said to be *computational zero knowledge* if the (computationally bounded) adversary $\mathcal{A}$ cannot distinguish between the simulation and runs of the real protocol.

We must show that, for any fixed random tape $\omega_{\mathcal{A}}$, it is computationally infeasible to distinguish the simulation from genuine runs of the protocol. This is exactly the computational assumption stated above, where $f(X)$ is the function $c(X, hist, \omega_{\mathcal{A}})$.

In other words, if the adversary $\mathcal{A}$ can distinguish between the simulation and the real protocol then it immediately solves the computational assumption. This completes the proof. □

In practice we conjecture that $\mathcal{K} = 136$ and $\mathcal{K}' = 51$ are sufficient to obtain 80 bits of security. The motivation for this choice is as follows. First, $3^{\mathcal{K}'} \approx 2^{80}$, so the probability of guessing the randomness used in line 12 of Algorithm 4 is negligible. Second, $3^{\Phi+0.63\mathcal{K}}/3^{\Phi+\mathcal{K}} = 3^{-0.37\mathcal{K}} \approx 2^{-80}$, where $3^{\Phi+0.63\mathcal{K}}$ comes from Heuristic 1 and $3^{\Phi+\mathcal{K}} = 3^{\mathcal{R}}$ is the total number of possibilities for r in $\mathcal{T}_{\mathcal{R}}$, so the probability of learning anything about sc from $r + sc$ seems to be negligible.

6.6.4 Soundness

The last task is to prove soundness of the scheme. We want to show that an adversary (which does not know the prover's private key s) is only able to impersonate the prover (i.e., she runs the protocol and is accepted by the verifier) with very small probability. To do this we will show that if an adversary is accepted by a verifier with non-small probability and if she is able to rewind the verifier, then she must know (or she can easily compute) the private key.

An adversary who does not know the private key can predict c with probability $\frac{1}{3^{\mathcal{C}l}}$. If she picks a random y, then compute $X = [y]P + [c]I$ and send X to the verifier, after receiving c', if $c' = c$, she answers with y and the verifier will accept. We call *probability of cheating* the probability of a dishonest prover being successfully accepted by the verifier.

Now, let us suppose that the adversary can do better than this, i.e., she can predict c with probability exceeding $\frac{1}{3^{cl}}$. We will use the adversary to solve the discrete log for the public key. The idea is to use the adversary to produce y_1 for a challenge c_1 on commitment X. We then repeat the experiment using the same randomness but a different challenge c_2, to obtain a response y_2 of the challenge c_2 on the *same* commitment X. We will show that if we can correctly answer to challenges $c_1 \neq c_2$ with $y_1 \neq y_2$, then we can recover the private key s.

Theorem 6.6.3 (Soundness). *An adversary who does not know the private key is only accepted by a verifier with very small probability.*

Proof. Let us suppose that an adversary $\mathcal{A}$ is accepted with probability $\varepsilon > \frac{1}{3^{cl}}$. Following the standard rewinding argument presented in [28, Appendix B] (inspired by [45]), we write $Succ(\omega_{\mathcal{A}}, c_1, c_2, \ldots, c_l) = true$ if an adversary $\mathcal{A}$, using random tape $\omega_{\mathcal{A}}$ is successfully identified by the verifier when successive challenges $c_1, c_2, \ldots, c_l$ are used.

We run the following algorithm:

step 1 Pick tuples c of l τ-adic expansions $(c_1, c_2, \ldots, c_l) \in \mathcal{T}_{\mathcal{C}}$ and run $\mathcal{A}$ with these challenges under a random tape $\omega_{\mathcal{A}}$ until $Succ(\omega_A, c) = true$. Call u the number of probes;

step 2 Try up to u l-tuples c' (different from c) and run $\mathcal{A}$ under the *same* random tape $\omega_{\mathcal{A}}$ until $Succ(\omega_{\mathcal{A}}, c') = true$. Output "fail" if a successful l-tuple c' is not found after u probes;

step 3 Output $\bar{y} = |y_j - y'_j|$ and $\bar{c} = |c'_j - c_j|$ where j is the first index such that $c_j \neq c'_j$ and y_j and y'_j are the corresponding correct responses of $\mathcal{A}$.

If the algorithm does not fail and such an adversary $\mathcal{A}$, given the same commitment X_j, is able to answer to two different challenges, c_j and c'_j with y_j and y'_j, it follows that $[y_j]P + [c_j]I = X_j = [y'_j]P + [c'_j]I$, and hence, $[(y_j - y'_j)]P = [(c'_j - c_j)]I$. Using the output of the above algorithm, the adversary can solve the DLP of I to the base P.

Note that as we are working with elliptic curves the curve order is known, so there is no difficulty with solving this equation. Roughly speaking, we can say that if an adversary is able to impersonate a prover than this is equivalent to say that she knows the prover's private key. □

6.7 τ-GPS vs. Standard GPS

We now briefly compare τ-GPS with the elliptic curve GPS variant of section 6.4. The standard GPS has lower bandwidth than τ-GPS and the online computations are simpler and we expect them to be faster for standard GPS (we did not evaluate the impact of τ-adic multiplication). The offline computations are quite efficient in both cases (especially when compared with GPS over an RSA modulus). A further advantage of standard GPS is that the security assurance is statistical zero knowledge rather than computational zero knowledge. The advantage of τ-GPS is that it does not require conversion between integers and Frobenius expansions.

6.8 Choosing the Right Protocol

We now give our recommendations on efficient identification protocols for constrained devices. Our suggestions are based on known results, commonly found in the Literature. For a more precise comparison, a implementation of the protocols is needed.

First, since the resources on the device are limited we recommend using Koblitz elliptic curves (rather than RSA moduli as proposed in [26, 27]). If there are no constraints on the offline computation time then we suggest using the standard GPS protocol or Girault-Lefranc. If the offline computation time is also limited then it is natural to improve the performance of the offline steps using Frobenius expansions. The precise choice of protocol then depends on the application:

- If bandwidth is the most precious resource then we recommend the Schnorr identification protocol.
- If computation time of the online stage is the most precious resource then we recommend the Girault-Lefranc method [27].
- If both bandwidth and computation time are precious then we recommend the standard GPS scheme [28].
- If code area and/or power consumption are the most precious resources then we recommend τ-GPS.

Table 6.6 summarises the above options.

Table 6.6: Choosing the right protocol.

Most precious resource	Our recommendation
Bandwidth	Schnorr identification protocol
Computation time of the online step	Girault-Lefranc method
Bandwidth and computation time	standard GPS scheme
Code area and/or power consumption	τ-GPS

6.9 Summary

In this chapter we presented the GPS identification scheme using τ-adic expansions. This speeds up the running time of the offline steps compared with standard GPS. We showed that cryptographic protocols can be made to operate with Frobenius expansions instead of integers, and this idea may have wider applications. The main advantage is that one does not need to convert between τ-adics and integers, saving code area and power consumption.

Chapter 7

Suggested Parameters

Contents

We analysed in Chapter 5 the complexity of attacks on variants of the Frobenius expansion DLP and in Chapter 6 the use of Frobenius expansions with GPS identification scheme. In this chapter we use this knowledge to determine parameters for efficient elliptic curve cryptography.

7.1 τ-DLP

We studied the τ-DLP in Chapter 5 and concluded that it can be solved with a low-memory algorithm in $\tilde{O}(2^{L/2})$ group operations. In the same chapter we determined the constant ε hidden in the $O(\ )$, so now we can calculate the minimum lengths required for the τ-adics to resist to τ-DLP algorithms.

We assume that the DLP is intractable if it needs more than 2^{78} group operations to solve it. When we have a general τ-adic or a τ-NAF, we have showed that we can use Gaudry-Schost algorithm to find a solution for an equivalent τ-adic $x + y\tau$, $|x| < a$ and $|y| < b$, to solve the τ-DLP in $2^{L/2+\varepsilon}$ group operations. Then we suggest the following lengths for a τ-adic to be resistant against τ-DLP algorithms:

- General τ-adic - When the proved values $a = 2.81\sqrt{2^L}$ and $b = 2\sqrt{2^L}$ are used, the expected value for ε is 4.06 so one can take a τ-adic of length $L \geqslant 148$. If we use the values for a and b considering the τ-adic distribution (i.e., $a = 2.28\sqrt{2^L}$ and $b = 1.8\sqrt{2^L}$), the expected value for ε is 3.82 so a τ-adic of length $L \geqslant 149$ is required;
- τ-NAF - The distribution of τ-NAFs show that the expected value for ε is 3.27, so one should take a τ-NAF of length $L \geqslant 150$;

When we have a Low Hamming weight τ-adic, we showed in Section 5.4 that the weight-w τ-DLP can be solved using van Oorschot and Wiener parallel collision search in $2^{w/2}\frac{7\binom{L/2}{w/2}^{3/2}}{\sqrt{w}}$ group operations. Taking $N = 2^w\binom{L}{w}$ we end up with $\tilde{O}(N^{3/4})$. Obviously, since the number of group operations also depends on the weight w, we cannot suggest a minimum bound for L.

7.2 Parameters Sizes for the τ-GPS

We presented the τ-GPS scheme in Chapter 6 and proved that it is computational zero-knowledge secure. Now, we analyse the sizes of the random τ-adic expansions used for representing the τ-GPS parameters such that the τ-GPS scheme be a zero knowledge proof of the private key.

7.2.1 Private Key s

We showed in Chapter 5 that one can use a square-root low-memory algorithm to solve the τ-DLP in $O(2^{\frac{\mathcal{S}}{2}})$ group operations. It is usual to choose s to be a τ-NAF, so we have fewer nonzeros and therefore, faster point multiplication. The Gaudry-Schost algorithm for τ-NAFs needs $2^{L/2+3.27}$ group operations to solve the DLP (see Section 5.6), hence we require $\mathcal{S} \geqslant 150$.

If enough memory is available, one can use BSGS over τ-NAF and solve the DLP in $\frac{8}{3}\sqrt{2^L}$ group operations. Hence, we require $\mathcal{S} \geqslant 158$. Note that using Pollard-rho using equivalence classes in $E(\mathbb{F}_{2^{163}})$ needs $\approx 2^{77.6}$ group operations to solve the DLP. Therefore, using τ-adic expansions of degree $\geqslant 150$ (or $\geqslant 158$ if enough memory is available) is no less secure than using integers.

7.2.2 Challenge c

From Section 6.6.4, we need $\mathcal{C}$ to be large enough such that the probability of cheating is very small. In practice, more than 2^{35} possibilities is enough to prevent an online attacker from guessing the right value for c.

Since c can be written as $x + y\tau$ for integers x and y and from Conjecture **??**, we have $|x| < 3\sqrt{2^{\mathcal{C}}}$ and $|y| < 2\sqrt{2^{\mathcal{C}}}$, we need $24(2^{\mathcal{C}}) \geqslant 2^{35}$ and therefore, $\mathcal{C} \geqslant 30.4$. Here, it does not make difference if we use the τ-adic distribution, since when the bounds for x and y are respectively $2.28\sqrt{2^L}$ and $1.8\sqrt{2^L}$, we need $16.416(2^{\mathcal{C}}) \geqslant 2^{35}$ and therefore, $\mathcal{C} \geqslant 30.9$, so we can take $\mathcal{C} \geqslant 31$ for both cases. If c is a τ-NAF, one should take $\mathcal{C} \geqslant 32$.

7.2.3 Commitment Source r

One can take $m = 163$ (see Section 6.6.1), $\mathcal{C} = 32$ (see Section 7.2.2), $\mathcal{S} = 150$ (see Section 7.2.1), $\mathcal{K} = 136$ and $\mathcal{K}' = 51$ (see Section 6.6.3). This gives $\mathcal{R} = \max\{\Phi + \mathcal{K}, m + \mathcal{K}'\} = \max\{321, 214\} = 321$ (recall that $\Phi = \mathcal{S} + \mathcal{C} + 3$).

7.3 Structure of the Random τ-adics Used in τ-GPS

We recall that point multiplication is very efficient when we use τ-adic expansions and since τ-NAF expansions have the minimum number of nonzero coefficients (see [4]), it is natural to choose random τ-NAFs to represent a parameter.

There are no restrictions with respect to the structure of the private key and the challenge, i.e., s and c can be a general τ-adic, a τ-NAF or a low-hamming weight τ-adic. We recommend to take s in non-adjacent form to improve performance in point multiplication with no loss of security.

We assumed in the zero knowledge part of the proof of security (Section 6.6.3), that the simulator picks γ uniformly and random in $\mathcal{T}_{\Phi,\mathcal{R}}$. Therefore, we cannot assume that r is a τ-NAF or r has low Hamming-weight, otherwise, the proof of security does not hold. Hence, we must use a general τ-adic to represent r.

We remark that for many cryptographic systems one would have to reprove the security if using random short τ-NAF expansions instead of random integers. For instance, it can be a bad idea to choose "random" values with some structure. In other words, the work of an attacker can become easier if he or she knows in advance that the τ-adic expansion is a τ-NAF for example.

7.4 τ-GPS Parameters

We summarise τ-GPS Parameters in Table 7.1 (compare with Table 6.1). The first number in each column is the value of the parameter and the second number (between parenthesis) is the number of bits needed to represent an element of the set (using $\frac{5L}{3}$ bits to represent a τ-adic of length L and $L+1$ bits if it can be chosen in non-adjacent form). We remark that, in principle, it should be possible to find better binary encodings for τ-adic expansions which require fewer than $\frac{5L}{3}$ bits.

Table 7.1: Numerical example of GPS scheme with τ-adic expansions.

$\mathcal{S}$	$\mathcal{C}$	$\mathcal{R}$
150 (151)	32 (33)	321 (535)

7.5 τ-GPS Using the Girault-Lefranc Idea

To speed up the computation of $s \times c$ we can use the Girault-Lefranc trick (see [27] for details). In other words, we insist that challenges c have at least $\mathcal{S} - 1$ zero coefficients between each pair of nonzero coefficients. It follows that computing $s \times c$ does not require Algorithm 4. This transforms the online step $y = r + sc$ into a single addition using Algorithm 4 with randomisation of the first $\mathcal{K}'$ coefficients.

In order to define $\mathcal{C}$ in this case we need to know how many τ-adic expansions with degree less than $\mathcal{C}$, Hamming weight w and at least $\mathcal{S} - 1$ zero coefficients between each pair of nonzero coefficients exist.

Theorem 7.5.1. *The number of possible τ-adic expansions of degree less than $\mathcal{C}$, Hamming weight w and at least $\mathcal{S} - 1$ zero coefficients between each pair of nonzero coefficients is:*

$$Z_{\mathcal{C},w,\mathcal{S}} = 2^w \Big[\binom{\mathcal{C} - w(\mathcal{S} - 1)}{w} + \sum_{i=1}^{\mathcal{S}-1} \binom{\mathcal{C} - i - (w-1)(\mathcal{S} - 1)}{w - 1} \Big] \tag{7.1}$$

Proof. We prove the theorem using combinatorics. For simplicity of notation, we call the nonzero coefficients "b", i.e. $b \in \{-1, +1\}$. We are required to have at least $\mathcal{S} - 1$ zero coefficients between each pair of nonzero coefficients; we use the word "block" for a configuration $[00 \cdots 0b]$ with exactly $\mathcal{S} - 1$ zeros before a nonzero b. Our problem is to count the number of ways to place w blocks of size $\mathcal{S}$ in $\mathcal{C}$ places. By viewing each block as a single bit our problem resumes to count the number of ways to place w blocks of size 1 in $\mathcal{C} - w(\mathcal{S} - 1)$ places.

We need to give a special attention to the leftmost block, which can be complete (i.e. with exactly $\mathcal{S}-1$ zeros before b) or not (i.e. with i zeros before b, where $0 \leq i \leq \mathcal{S}-2$). So, we divide our problem into two parts:

1. Leftmost block complete.

   We want to locate w blocks (remember, each block with $\mathcal{S}-1$ zeros before b) in $\mathcal{C}-w(\mathcal{S}-1)$ places. So, we have $\binom{\mathcal{C}-w(\mathcal{S}-1)}{w}$. As b can be either -1 or $+1$, we have:

$$2^w \binom{\mathcal{C}-w(\mathcal{S}-1)}{w} \tag{7.2}$$

2. Leftmost nonzero coefficient in position i from the left, where $i \in \{1, \mathcal{S}-1\}$.

   Now, our problem is to locate $w-1$ blocks in the remaining $\mathcal{C}-i$ places. As each block was reduced from size $\mathcal{S}$ to size 1, our free space now has $\mathcal{C}-i-(w-1)(\mathcal{S}-1)$ places. So, we have:

$$2^w \sum_{i=1}^{\mathcal{S}-1} \binom{\mathcal{C}-i-(w-1)(\mathcal{S}-1)}{w-1}. \tag{7.3}$$

Now, just add (7.2) and (7.3) and the proof is complete. □

Hence, the total number of possible τ-adic expansions of degree less than $\mathcal{C}$ and with at least $\mathcal{S}-1$ zero coefficients between each pair of non zero coefficients is

$$Z_{\mathcal{C},\mathcal{S}} = \sum_{w=1}^{\lfloor \frac{\mathcal{C}+\mathcal{S}-1}{\mathcal{S}} \rfloor} 2^w \left[\binom{\mathcal{C}-w(\mathcal{S}-1)}{w} + \sum_{i=1}^{\mathcal{S}-1} \binom{\mathcal{C}-i-(w-1)(\mathcal{S}-1)}{w-1} \right] \tag{7.4}$$

It follows (see Table 7.2 and compare with Table 6.1) that if $\mathcal{S}=150$ then we need $\mathcal{C}=759$ to get $Z_{\mathcal{C},\mathcal{S}} > 2^{35}$. This results in $\mathcal{R} = \max\{\Phi+\mathcal{K}, m+\mathcal{K}'\} =$

$\max\{1048, 214\} = 1048$. Note that one can transmit/store c using much fewer than 760 bits. However, r and y are not sparse, so would need around 1747 bits to represent.

Table 7.2: Parameters for the Girault-Lefranc variant of $\tau-$GPS.

$\mathcal{S}$	$\mathcal{C}$	$\mathcal{R}$
150 (151)	759 (760)	1048 (1747)

Now we have a fast online computation of $c \times s$, at the cost of a very large y to be transmitted, which can be a serious drawback due to memory restrictions. Although c is also very large, it has very few nonzero coefficients, and therefore, the bit representation of c can be much more efficient than simply $\mathcal{C} + 1$ bits, if the verifier transmit c by using only the nonzero positions. However as y is expected to be dense, the same cannot be done when sending y.

7.5.1 Performance analysis

We give more detail about how to efficiently compute each of the three steps of the GPS protocol.

Step 1: We are supposed to choose a random element $r \in \mathcal{T}_{\mathcal{R}}$ (where $\mathcal{R}$ may be 321 or 1048) and compute $X = [r]P$. Note that $\mathcal{R}$ is larger than m. In practice it would be more efficient to reduce r modulo $\tau^m - 1$ before performing the computation of X (this may lead to coefficients outside the set $\{-1, 0, 1\}$).

A more efficient alternative is as follows. Choose a random $r' \in \mathcal{T}_m$ and compute $X = [r']P$. It is only necessary to store r', requiring $\approx 5m/3$ bits. Indeed, one could choose r' in non-adjacent form, in which case at most $m + 1$ bits are required.

When the long value r is needed in the online step we can simply choose the coefficients r_j randomly subject to the constraint that for $0 \leqslant i < m$ we have

$$\sum_{j=0}^{\lfloor(\mathcal{R}-i)/m\rfloor} r_{jm+i} = r'_i$$

which can be done efficiently. Note that this variant is not covered by our security analysis since not all possible values of $r \in \mathcal{T}_{\mathcal{R}}$ can arise. A security analysis of this case is a topic for future research.

Step 2: The computation of $r + sc$ is done using repeated calls to Algorithm 4. One could also use the Karatsuba idea to speed-up the polynomial multiplication. For more details of improvements see [27].

Step 3: To compute $[y]P + [c]I$ one can reduce y (and c if necessary) modulo $\tau^m - 1$. One can also use the standard multiexponentiation and precomputation techniques to speed this up (see [2]).

7.6 Summary

In this chapter we gave some suggestions of parameters sizes based on the results of Chapters 5 and 6. We also presented some efficient ways to compute each of the three steps of the GPS protocol.

Chapter 8

Conclusion and Future Paths

This is the end of our journey, and it is time to look back of what we have done. The job here is simplified because we have already summarised each chapter at its end. But now we emphasize the main ideas.

As we had said before, we started this project proposing a problem ("Can we use shorter parameters if we run the GPS identification protocol with Frobenius expansions in the place of integers ?"). To answer this question, we needed first to study Koblitz curves and Frobenius expansions, to know how they behave. So we did that in Chapter 4. Then, a natural question arose: "Are there low-memory algorithms for the τ-DLP ?". The response came with Chapter 5, which also solved our first question, i.e., since there are low-memory algorithms for the τ-DLP, we cannot use small parameters with the τ-GPS. However, the τ-GPS protocol yet has some attractiveness, being useful to applications with limited offline computation time and limited code area. So we studied the τ-GPS in Chapter 6 and gave a security analysis for the protocol using τ-adics.

8.1 Future Paths

Inspired by a famous song: "And now, the end is near and so I face the final curtain" we can look further and pose some future paths for our journey.

- We proved bounds for integers x and y when a τ-adic with coefficients in $\{-1, 0, 1\}$ is mapped to $x + y\tau$ and conjecture that we can shorten those bounds using the τ-adic distribution.

  This leads to three open problems:

  - prove the *exact* bounds for x and y, which would open the doors for more efficient low-memory algorithms;
  - understand better the pictures that show the τ-adic distribution.
  - generalise this approach and determine the sizes of bounds a' and b' for integers $x \leqslant a'$ and $y \leqslant b'$, when working with a general coefficient set;

- We showed that the weight-w τ-DLP can be solved using van Oorschot and Wiener parallel collision search in expected time $O(N^{3/4})$ for some measure N of the size of the problem. We leave as a challenge to algorithm designers to 'close the gap' with time/memory tradeoff algorithms and propose a low-memory algorithm which solves the weight-w τ-DLP in expected time $O(N^{1/2})$;

- We proved that our τ-GPS protocol is computational zero knowledge. We left as an open problem to modify the τ-GPS scheme in order to make it statistical or even perfect zero knowledge.

Naturally, the suggestions for future paths are not limited by this list, since Frobenius expansions have a large potential to be explored. We hope that this thesis can draw attention to the use of τ-adic expansions in Cryptography, and many other applications can derive from that.

References

[1] R. Avanzi, M. Ciet, and F. Sica. Faster Scalar Multiplication on Koblitz Curves Combining Point Halving with the Frobenius Endomorphism. In *Public Key Cryptography, PKC 2004*, pages 28–40, 2004.

[2] R. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, volume 34 of *Discrete Mathematics and Its Applications*. CRC Press, 2005.

[3] R. Avanzi, V. Dimitrov, C. Doche, and F. Sica. Extending Scalar Multiplication Using Double Bases. In X. Lai and K. Chen, editors, *ASIACRYPT 2006 - 12th International Conference on the Theory and Application of Cryptology and Information Security*, volume 4284 of *Lectures Notes in Computer Science*, pages 130–144, Shangai, China, December 2006. Springer.

[4] R. Avanzi, C. Heuberger, and H. Prodinger. Minimality of the Hamming Weight of the τ-NAF for Koblitz Curves and Improved Combination with Point Halving. In B. Preneel and S. Tavares, editors, *Selected Areas in Cryptography, 12th International Workshop, SAC 2005*, volume 3897 of *Lectures Notes in Computer Science*, Kingston, ON, Canada, August, 11–12 2005. Springer Berlin / Heidelberg.

[5] R. Avanzi, H. Prodinger, and C. Heuberger. Scalar Multiplication on Koblitz Curves Using the Frobenius Endomorphism and its Combination with Point Halving: Extensions and Mathematical Analysis. In *Algorithmica Special Issue on Analysis of Algorithms*, volume 46 of *Algorithmica*, pages 249–270, 2006.

[6] R. Avanzi and F. Sica. Scalar Multiplication on Koblitz Curves Using Double Bases. In P. Nguyen, editor, *Vietcrypt 2006 - First International Conference on Cryptology in Vietnam*, volume 4341 of *Lectures Notes in Computer Science*, pages 131–146, Hanoi, Vietnam, September 2006. Springer.

[7] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[8] W. Benits and S. Galbraith. The GPS Identification Scheme Using Frobenius Expansions, 2007. To appear in WEWOrC 2007 - LCNS 4945.

[9] I. Blake, G. Seroussi, and N. Smart, editors. *Advances in Elliptic Curve Cryptography*, volume 317 of *London Mathematical Society.* Cambridge University Press, New York, NY, USA, 2005.

[10] W. Bosma, J. Cannon, and C. Playoust. The MAGMA Algebra System I: The User Language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997.

[11] R. Brent. An improved Monte Carlo factorization algorithm. *BIT Numerical Mathematics*, 20:176–184, 1980.

[12] B. Brumley and K. Järvinen. Koblitz Curves and Integer Equivalents of Frobenius Expansions. In C. Adams, A. Miri, and M. Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop—SAC '07*, volume 4876 of *Lecture Notes in Computer Science*, pages 126–137. Springer-Verlag, 2007.

[13] P. Cameron. *Combinatorics - Topics, Techniques, Algorithms.* Cambridge University Press, 1994.

[14] J.-H. Cheon and H.-T. Kim. Analysis of Low Hamming Weight Products. To appear in Discrete Appl. Math.

[15] E. G. Coffman, P. Flajolet, L. Flatto, and M. Hofri. The Maximum of a Random Walk and Its Application to Rectangle Packing. *Probability in Engineering and Informational Sciences*, 12:373–386, 1998.

[16] H. Cohen. *A Course in Computational Algebraic Number Theory.* Graduate Texts in Mathematics. Springer-Verlag, New York, NY, USA, 1993.

[17] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 2nd edition, 2001.

[18] J. Coron, D. M'Raïhi, and C. Tymen. Fast Generation of Pairs $(k, [k]P)$ for Koblitz Elliptic Curves. In S. Vaudenay and M. Youssef, editors, *Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001*, volume 2259 of *Lecture Notes in Computer Science*, pages 151–164, Toronto, Ontario, Canada, August 2001. Springer-Verlag.

[19] J.S. Coron, D. Lefranc, and G. Poupard. A New Baby-Step Giant-Step Algorithm and Some Applications to Cryptanalysis. In J. Rao and B Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop*, volume 3659 of *Lectures Notes in Computer Science*, pages 47–60, Edinburgh, UK, August 29 - September 1 2005. Springer Berlin / Heidelberg.

[20] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[21] N. Ebeid and M. Hasan. On τ-adic Representations of Integers. *Designs, Codes and Cryptography*, 45(3):271–296, 2007.

[22] U. Feige, A. Fiat, and A. Shamir. Zero-Knowledge Proofs of Identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[23] S. Galbraith. Elliptic Curve Cryptography. Available in: http://www.isg.rhul.ac.uk/ sdg/ecc.html.

[24] R. Gallant, R. Lambert, and S. Vanstone. Improving the Parallelized Pollard Lambda Search on Anomalous Binary Curves. *Mathematics of Computation*, 69(232):1699–1705, 2000.

[25] P. Gaudry and E. Schost. A Low-Memory Parallel Version of Matsuo, Chao and Tsujii's Algorithm. In D. Buel, editor, *Algorithmic Number Theory, 6th International Symposium, ANTS-VI*, volume 3076 of *Lectures Notes in Computer Science*, pages 208–222, Burlington, VT, USA, June, 13–18 2004. Springer.

[26] M. Girault. Self-Certified Public Keys. In D.W. Davies, editor, *Advances in Cryptology – EUROCRYPT 1991, Workshop on the Theory and Application of of Cryptographic Techniques*, volume 547 of *Lectures Notes in Computer Science*, pages 490–497, Brighton, UK, April, 8–11 1992. Springer-Verlag.

[27] M. Girault and D. Lefranc. Public Key Authentication with One (Online) Single Addition. In M. Joye and J.J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop*, Lectures Notes in Computer Science, pages 413–427, Cambridge, MA, USA, August 11–13 2004. Springer.

[28] M. Girault, G. Poupard, and J. Stern. On the Fly Authentication and Signature Schemes Based on Groups of Unknown Order. *Journal of Cryptology*, 19(4):463–487, October 2006.

[29] O. Goldreich. Zero-Knowledge Twenty Years After its Invention. Available in: citeseer.ist.psu.edu/goldreich02zeroknowledge.html, 2002.

[30] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[31] C. Günther, T. Lange, and A. Stein. Speeding up the Arithmetic on Koblitz Curves of Genus Two. In D. R. Stinson and S. E. Tavares, editors, *Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Proceedings*, volume 2012 of *Lecture Notes in Computer Science*, pages 106–117, Waterloo, Ontario, Canada, August, 14–15 2000. Springer.

[32] J.Hoffstein and J. Silverman. Random Small Hamming Weight Products with Applications to Cryptography. *Discrete Applied Mathematics*, 130(1):37–49, 2003.

[33] N. Koblitz. CM-Curves with Good Cryptographic Properties. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference*, volume 576 of *Lectures Notes in Computer Science*, pages 279–287, London, UK, August, 11–15 1992. Springer-Verlag.

[34] T. Lange. Koblitz Curve Cryptosystems. *Finite Fields and Their Applications*, 11, Issue 2:200–229, April 2005.

[35] T. Lange and I. Shparlinski. Collisions in Fast Generation of Ideal Classes and Points on Hyperelliptic and Elliptic Curves. In *Applicable Algebra in Engineering, Communication and Computing*, volume 15, pages 329–337. Springer, 2005.

[36] W. Meier and O. Staffelbach. Efficient Multiplication on Certain Nonsupersingular Elliptic Curves. In E. Brickell, editor, *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference*, volume Volume 740 of *Lectures Notes in Computer Science*, page 333, Santa Barbara, California, USA, August, 16–20 1992. Springer Berlin / Heidelberg.

[37] A. Menezes, P van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 2001.

[38] A. Menezes and S. A. Vanstone. Elliptic Curve Cryptosystems and their Implementation. *Journal of Cryptology*, 6:209–224, 1993.

[39] F. Mosteller, R. E. K. Rourke, and G. B. Thomas. *Probability and Statistics.* Reading, MA, 1961.

[40] T. Okamoto, H. Katsuno, and E. Okamoto. A Fast Signature Scheme Based on New Online Computation. In J. S. Sichman, F. Bousquet, and P. Davidsson, editors, *Multi-Agent-Based Simulation, Third International Workshop, MABS 2002*, volume 2581 of *Lectures Notes In*

Computer Science, pages 111–121, Bologna, Italy, July, 15–16 2003. Springer.

[41] J. Pollard. Monte Carlo Methods for Index Computation mod p. *Mathematics of Computation*, 32:918–924, 1978.

[42] J. Pollard. Kangaroos, Monopoly and Discrete Logarithms. *Journal of Cryptology*, 13(4):437–447, December 2000.

[43] G. Poupard and J. Stern. Security Analysis of a Practical "on the fly" Authentication and Signature Generation. In K. Nyberg, editor, *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques*, volume 1403 of *Lecture Notes in Computer Science*, pages 422–436, Espoo, Finland, May 31 - June 4 1998. Springer.

[44] G. Qiao and K.-Y. Lam. RSA Signature Algorithm for Microcontroller Implementation. In J.-J. Quisquater and B. Schneier, editors, *Smart Card Research and Applications, This International Conference, CARDIS '98*, volume 1820 of *Lectures Notes in Computer Science*, pages 353–356, Louvain-la-Neuve, Belgium, September, 14–16 2000. Springer.

[45] C. P. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[46] C.P. Schnorr. Efficient Identification and Signatures for Smart Cards. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference*, volume 435 of *Lecture Notes in Computer Science*, pages 235–251, Santa Barbara, California, USA, August, 20–24 1990. Springer Verlag.

[47] R. Sedgewick, T. Szymanski, and A. Chi-Chih Yao. The complexity of finding cycles in periodic functions. *SIAM Journal of Computation*, 11(2):376–390, 1982.

[48] N. Smart. Elliptic Curve Cryptosystems over Small Fields of Odd Characteristic. *Journal of Cryptology*, 12(2):141–151, 1999.

[49] N. Smart. *Cryptography: An Introduction.* McGraw-Hill, December 2004.

[50] J. Solinas. An Improved Algorithm for Arithmetic on a Family of Elliptic Curves. In B Kaliski Jr, editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference*, volume 1294 of *Lectures Notes in Computer Science*, pages 357–371, Santa Barbara, California, USA, August, 17–21 1997. Springer-Verlag.

[51] J. Solinas. Efficient Arithmetic on Koblitz Curves. *Designs Codes and Cryptography*, 19(2-3):195–249, 2000.

[52] W. Stallings. *Cryptography and Network Security: Principles and Practice.* Prentice Hall, third edition, 2002.

[53] D. Stinson. Some Baby-Step Giant-Step Algorithms for the Low Hamming Weight Discrete Logarithm Problem. *Mathematics of Computation*, 71(237):379–391, 2002.

[54] D. Stinson. *Cryptography Theory and Practice.* Discrete Mathematics and its Applications. Chapman & Hall, third edition, 2006.

[55] E. Teske. Square-root Algorithms for the Discrete Logarithm Problem (A Survey). In W. Gruyter, editor, *Public-key Cryptography and Computational Number Theory*, pages 283–301, 2001.

[56] Edlyn Teske. Computing discrete logarithms with the parallelized kangaroo method. *Discrete Applied Mathematics*, 130(1):61–82, 2003.

[57] P. van Oorschot and M. Wiener. Improving Implementable Meet-In-The-Middle Attacks by Orders of Magnitude. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference*, volume 1109 of *Lectures Notes in Computer Science*, pages 229–236, Santa Barbara, California, USA, August, 18–22 1996. Springer.

[58] P. van Oorschot and M. Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[59] E. Weisstein. Random Walk–1-Dimensional - From MathWorld - A Wolfram Web Resource. Available in: http://mathworld.wolfram.com/RandomWalk1-Dimensional.html.

[60] M. J. Wiener and R. J. Zuccherato. Faster Attacks on Elliptic Curve Cryptosystems. In S. Tavares and H. Meijer, editors, *Selected Areas in Cryptography '98, SAC'98*, volume 1556 of *Lectures Notes In Computer Science*, pages 190–200, Kingston, Ontario, Canada, August, 17–18 1999. Springer.

Index