Chapter 11

Basic Algorithms for Algebraic Groups

In Section 4.1 a number of basic computational tasks for an algebraic group G were listed. Some of these topics have been discussed already, especially providing efficient group operations and compact representations for group elements. But some other topics (such as efficient exponentiation, generating random elements in G and hashing from or into G) require further attention. The goal of this chapter is to briefly give some details about these tasks for the algebraic groups of most interest in the book.

The main goal of the chapter is to discuss exponentiation and multi-exponentiation. These operations are crucial for efficient discrete logarithm cryptography and there are a number of techniques available for specific groups that give performance improvements.

It is beyond the scope of this book to present a recipe for the best possible exponentiation algorithm in a specific application. Instead, our focus is on explaining the mathematical ideas that are used. For an "implementors guide" in the case of elliptic curves we refer to Bernstein and Lange [53].

Let G be a group (written in multiplicative notation). Given $g \in G$ and $a \in \mathbb{N}$ we wish to compute g^a. We assume in this chapter that a is a randomly chosen integer of size approximately the same as the order of g, and so a varies between executions of the exponentiation algorithm. If g does not change between executions of the algorithm then we call it a **fixed base** and otherwise it is a **variable base**.

As mentioned in Section 2.8, there is a significant difference between the cases where g is fixed (and one is computing g^a repeatedly for different values of a) and the case where both g and a vary. Section 2.8 already briefly mentioned addition chains and sliding window methods. The literature on addition chains is enormous and we do not delve further into this topic. Window methods date back to Brauer in 1939 and sliding windows to Thurber; we refer to Bernstein's excellent survey [45] for historical details. Other references for fast exponentiation are Chapters 9 and 15 of [16], Chapter 3 of [274]

and Sections 14.6 and 14.7 of [418].

11.1 Efficient Exponentiation Using Signed Exponents

In certain algebraic groups, computing the inverse of a group element is much more efficient than a general group operation. For example, Exercise 6.3.1 and Lemma 6.3.12 show that inversion in $G_{q,2}$ and $\mathbb{T}_2(\mathbb{F}_q)$ is easy. Similarly, inversion in elliptic and hyperelliptic curve groups is easy (see Section 9.1 and Exercises 10.4.2 and 10.4.17). Hence, one can exploit inversion when computing exponentiation and it is desirable to consider signed expansions for exponents.

Signed expansions and addition-subtraction chains have a long history.[1] Morain and Olivos [438] realised that, since inversion is easy for elliptic curve groups, signed expansions are natural in this context.

11.1.1 Non-Adjacent Form

We now discuss the non-adjacent form (NAF) of an integer a. This is the best signed expansion in the sense that it has the minimal number of non-zero coefficients and can be computed efficiently. The non-adjacent form was discussed by Reitwiesner [499] (where it is called "property M"). Reitwiesner proved that the NAF is unique, has minimal weight among binary expansions with coefficients in $\{-1, 0, 1\}$, and he gave an algorithm to compute the NAF of an integer. These results have been re-discovered and simplified numerous times (we refer to Section IV.2.4 of [64] and Section 5 of [544] for references).

Definition 11.1.1. Let $a \in \mathbb{N}$. A representation $a = \sum_{i=0}^{l} a_i 2^i$ is a **non-adjacent form** or **NAF** if $a_i \in \{-1, 0, 1\}$ for all $0 \leq i \leq l$ and $a_i a_{i+1} = 0$ for all $0 \leq i < l$. If $a_l \neq 0$ then the **length** of the NAF is $l + 1$.

One can transform an integer a into NAF representation using Algorithm 6. This is a "right-to-left" algorithm in the sense that it processes the least significant bits first. We define the operator $a \pmod{2m}$ to be reduction of a modulo $2m$ to the range $\{-m+1, \ldots, -1, 0, 1, \ldots, m\}$. In particular, if a is odd then $a \pmod 4 \in \{-1, 1\}$. An alternative right-to-left algorithm is given in the proof of Theorem 11.1.12.

Exercise 11.1.2. Prove that Algorithm 6 outputs a NAF.

Example 11.1.3. We compute the NAF representation of $a = 91$. Since $91 \equiv 3 \pmod 4$ the first digit is -1, which we denote as $\overline{1}$. Note that $2^2 \| 92$ so the next digit is 0. Now, $92/4 = 23 \equiv 3 \pmod 4$ and the next digit is $\overline{1}$. Since $2^3 \| 24$ the next 2 digits are 0. Continuing one finds the expansion to be $10\overline{1}00\overline{1}0\overline{1}$.

Lemma 11.1.4 shows that a simple way to compute a NAF of an integer a is to compute the binary representation of $3a$, subtract the binary representation of a (writing the result in signed binary expansion, in other words, performing the subtraction without carries), and discard the least significant bit. We write this as $((3a) - a)/2$.

Lemma 11.1.4. *Let $a \in \mathbb{N}$. Then the signed binary expansion $((3a) - (a))/2$ is in non-adjacent form.*

[1] Reitwiesner's long paper [499] suggests signed expansions as a way to achieve faster arithmetic (e.g., multiplication and division) but does not discuss exponentiation. Brickell, in 1982, seems to have been the first to suggest using negative powers of g to speed up the computation of g^a; this was in the context of computing $m^e \pmod N$ in RSA and required the precomputation of $m^{-1} \pmod N$.

Algorithm 6 Convert an integer to non-adjacent form

INPUT: $a \in \mathbb{N}$
OUTPUT: $(a_l \dots a_0)$
1: $i = 0$
2: **while** $a \neq 0$ **do**
3: **if** a even **then**
4: $a_i = 0$
5: **else**
6: $a_i = a \pmod 4$
7: **end if**
8: $a = (a - a_i)/2$
9: $i = i + 1$
10: **end while**
11: **return** $a_l \dots a_0$

Proof: Write a in binary as $(a_l \dots a_0)_2$ and write $3a$ in binary as $(b_{l+2} \dots b_0)_2$. Set $a_{-1} = a_{l+1} = a_{l+2} = 0$ and $c_{-1} = 0$. Then $b_i = a_i + a_{i-1} + c_{i-1} - 2c_i$ where $c_i = \lfloor (a_i + a_{i-1} + c_{i-1})/2 \rfloor \in \{0, 1\}$ is the carry from the i-th addition.

Now consider the signed expansion $s_i = b_i - a_i \in \{-1, 0, 1\}$. In other words, $s_i = a_{i-1} + c_{i-1} - 2c_i$. Clearly $b_0 = a_0$ and so $s_0 = 0$. We show that $s_i s_{i+1} = 0$ for $1 \leq i \leq l+1$. Suppose i is such that $s_i \neq 0$. Since $a_{i-1} + c_{i-1} \in \{0, 1, 2\}$ and $a_{i-1} + c_{i-1} \equiv s_i \pmod 2$ it follows that $a_{i-1} + c_{i-1} = 1$. This then implies that $c_i = \lfloor (1 + a_i)/2 \rfloor = a_i$. Hence, $c_{i+1} = a_i$ and $s_{i+1} = a_i + c_i - 2c_{i+1} = 0$. □

Example 11.1.5. Taking $a = 91$ again, we have $3 \cdot 91 = 273 = (100010001)_2$. Computing $(3a) - a$ is

$$100010001 \; - \; 1011011 \; = \; 10\bar{1}00\bar{1}0\bar{1}0.$$

Exercise 11.1.6. Compute NAFs for $a = 100, 201, 302$ and 403.

We now state and prove some properties of NAFs.

Exercise 11.1.7. Show that if $a_l \dots a_0$ is a NAF of a then $(-a_l) \dots (-a_0)$ is a NAF of $-a$.

Lemma 11.1.8. *The NAF representation of $a \in \mathbb{Z}$ is unique.*

Proof: Without loss of generality we may assume $a > 1$. Note that $a = 1$ has a unique representation as a NAF, so assume $a > 1$. Let $a \in \mathbb{N}$ be the smallest positive integer such that a has two (or more) distinct representations as a NAF, call them $\sum_{i=0}^{l} a_i 2^i$ and $\sum_{i=0}^{l'} a'_i 2^i$. If a is even then $a_0 = a'_0 = 0$ and so we have two distinct NAF representations of $a/2$, which contradicts the minimality of a. If a is odd then $a \equiv \pm 1 \pmod 4$ and so $a_0 = a'_0$ and $a_1 = a'_1 = 0$. Hence, we obtain two distinct NAF representations of $(a - a_0)/4 < a$, which again contradicts the minimality of a. □

Exercise 11.1.9.★ Let $a \in \mathbb{N}$. Show that a has a length $l + 1$ NAF representation if and only if $2^l - d_l \leq a \leq 2^l + d_l$ where

$$d_l = \begin{cases} (2^l - 2)/3 & \text{if } l \text{ is odd} \\ (2^l - 1)/3 & \text{if } l \text{ is even.} \end{cases}$$

Also show that if $a > 0$ then $a_l = 1$ and prove that the length of a NAF is at most one more than the length of the binary expansion of a.

Definition 11.1.10. Let $\mathcal{D} \subset \mathbb{Z}$ be such that $0 \in \mathcal{D}$. The **weight** of a representation $a = \sum_{i=0}^{l} a_i 2^i$ where $a_i \in \mathcal{D}$ is the number of values $0 \leq i \leq l$ such that $a_i \neq 0$. The weight of a is denoted weight(a). The **density** of the representation is weight$(a)/(l+1)$.

Exercise 11.1.11. Show that if $a \in \mathbb{N}$ is uniformly chosen in $2^l \leq a < 2^{l+1}$ and represented using the standard binary expansion then the expected value of the weight is $(l+1)/2$ and therefore the expected value of the density is $1/2$.

Theorem 11.1.12. *The NAF of an integer $a \in \mathbb{N}$ has minimal weight among all signed expansions $a = \sum_{i=0}^{l} a_i 2^i$ where $a_i \in \{-1, 0, 1\}$.*

Proof: Let $a = \sum_{i=0}^{l} a_i 2^i$ where $a_i \in \{-1, 0, 1\}$ be any signed expansion of a. Perform the following string re-writing process from right to left (i.e., starting with a_0). If $a_i = 0$ or $a_{i+1} = 0$ then do nothing. Otherwise (i.e., $a_i \neq 0$ and $a_{i+1} \neq 0$) there exists an integer $k \geq 1$ such that the sequence $a_{i+k}a_{i+k-1} \dots a_i a_{i-1}$ is of the form

$$01 \dots 10, \qquad \overline{1}1 \dots 10, \qquad 1\overline{1} \dots \overline{1}0, \qquad 0\overline{1} \dots \overline{1}0.$$

In each case replace the pattern with the following

$$10 \dots 0\overline{1}0, \qquad 0 \dots 0\overline{1}0, \qquad 0 \dots 010, \qquad \overline{1}0 \dots 010.$$

In each case, the resulting substring has weight less than or equal to the weight of the previous substring and is in non-adjacent form (at least up to $a_{i+k-1}a_{i+k} = 0$). Continuing the process therefore yields a NAF expansion of a of weight less than or equal to the weight of the original signed expansion. □

Example 11.1.13. We re-compute the NAF representation of $a = 91$, using the method in the proof of Theorem 11.1.12. First note that the binary expansion of 91 is 1011011. One replaces the initial 011 by $10\overline{1}$ to get $101110\overline{1}$ One then replaces 0111 by $100\overline{1}$. Continuing one determines the NAF of 91 to be $10\overline{1}00\overline{1}0\overline{1}$.

We have established that the NAF of an integer a is unique, has minimal weight, and has length at most one bit more than the binary expansion of a. Finally, we sketch a probabilistic argument that shows that the density of a NAF is expected to be $1/3$.

Lemma 11.1.14. *Let $l \in \mathbb{N}$. Define d_l to be the expected value of the density of the NAF representation of uniformly chosen integers $2^{l+1}/3 < a < 2^{l+2}/3$. Then d_l tends to $1/3$ as l goes to infinity.*

Proof: (Sketch) Write $a = \sum_{i=0}^{l} a_i 2^l$ for the NAF representation of $a \in \mathbb{N}$. Note that Algorithm 6 has the property that, if $a_i \neq 0$, then $a_{i+1} = 0$ but the value of a_{i+2} is independent of the previous operations of the algorithm. Hence, if the bits of a are considered to be chosen uniformly at random then the probability that $a_{i+2} \neq 0$ is $1/2$. Similarly, the probability that $a_{i+2} = 0$ but $a_{i+3} \neq 0$ is $1/4$, and so on. Hence, the expected number of zeroes after the non-zero a_i is (at least approximately, since the expansion is not infinite)

$$E = 1 \cdot \tfrac{1}{2} + 2 \cdot \tfrac{1}{4} + 3 \cdot \tfrac{1}{8} + \cdots = \sum_{i=1}^{\infty} \tfrac{i}{2^i}.$$

Now,

$$E = 2E - E = \sum_{i=1}^{\infty} \tfrac{i}{2^{i-1}} - \sum_{i=1}^{\infty} \tfrac{i}{2^i} = 1 + \sum_{j=1}^{\infty} \tfrac{1}{2^j} = 2.$$

Hence, on average, there are two zeros between adjacent non-zero coefficients and the density tends to $1/3$. □

Exercise 11.1.15. Prove that the number of distinct NAFs of length k is $(2^{k+2} - (-1)^k)/3$.

Exercise 11.1.16.★ Write down an algorithm to list all NAFs of length k.

For some applications it is desired to compute a low-density signed expansion from left to right; Joye and Yen [321] give an algorithm to do this.

11.1.2 Width-w Non-Adjacent Form

Definition 11.1.17. Let $w \in \mathbb{N}_{\geq 2}$ and $m = 2^{w-1} - 1$. Define $\mathcal{D} = \{0\} \cup \{a \in \mathbb{Z} : a \text{ odd}, |a| \leq m\} = \{0, \pm 1, \pm 3, \ldots, \pm(2^{w-1} - 1)\}$. A representation $\sum_{i=0}^{l} a_i 2^i$ is a **width-w non-adjacent form** (also written w-NAF or NAF_w) if $a_i \in \mathcal{D}$ and if $a_i \neq 0$ implies $a_{i+1} = \cdots = a_{i+w-1} = 0$. When writing such expansions we write $\overline{n}$ for the digit $-n$. If $a_l \neq 0$ then the **length** of the w-NAF is $l + 1$.

This notion was first proposed in Miyaji, Ono and Cohen [430] and rediscovered in Section IV.2.5 of Blake, Seroussi and Smart [64] and Section 3.2 of Solinas [576] (the latter paper gives us the terminology).

Example 11.1.18. The NAF of an integer $a \in \mathbb{N}$ is a width-2 NAF.

All the results and proofs given above regarding NAFs generalise immediately to the case of w-NAFs.

Exercise 11.1.19. Change line 7 of Algorithm 6 to read $a_i = a \pmod{2^w}$. Show that this algorithm computes a w-NAF of the integer a.

Example 11.1.20. We compute the 3-NAF of 151. Since $151 \equiv 7 \pmod 8$ we have $a_0 = -1$. One then finds $a_1 = a_2 = 0$. Now $(151 + 1)/8 = 19 \equiv 3 \pmod 8$ so $a_3 = 3$. Finally $a_4 = a_5 = a_6 = 0$ and $a_7 = 1$. The 3-NAF of 151 is therefore $1000300\overline{1}$, having weight 3. In comparison, the NAF of 151 is $1010\overline{1}00\overline{1}$, which has weight 4.

Exercise 11.1.21. Prove that the w-NAF is unique.

Exercise 11.1.22. Write pseudocode for an efficient left-to-right exponentiation algorithm using the w-NAF representation of the exponent a. Show that the precomputation requires one squaring and $2^{w-2} - 1$ multiplications.

Exercise 11.1.23. Prove that the length of a w-NAF of an integer $a \in \mathbb{N}$ is at most one more than the bit-length of a.

Exercise 11.1.24. Modify the proof of Lemma 11.1.14 to show that the expected density of the width-w NAF of a randomly chosen l-bit integer tends to $1/(w + 1)$ as l goes to infinity.

The length of a w-NAF expansion of a can be less than $\log_2(a)$, since the most significant coefficient can be as big as $2^{w-1} - 1$. Intuitively, one might expect the length on average to be approximately $\log_2(a) - (w - 1)/2$. Further discussion of this issue together with a precise analysis of the density of w-NAFs is given by Cohen [137].

It is shown in Theorem 2.3 of Avanzi [17] and Theorem 3.3 of Muir and Stinson [442] that the w-NAF has minimal weight among all signed expansions with that digit set. Analogues of the w-NAF that can be computed from left-to-right have been studied in a number of papers, starting in Section 3 of [17] and [441].

11.1.3 Further Methods

There are two other ways to exploit these ideas, namely fractional window NAFs (proposed by Möller [432, 433]) and (fractional) sliding windows over NAFs.[2] The sliding window algorithms are natural analogues of the ones mention in Example 2.8.5 and Exercise 2.8.6.

The point of these methods is to allow fine-tuning of the precomputation cost so that one can minimise the expected overall computation for exponents of a given bit-length. The basic idea is to precompute fewer group elements than the standard version of the width-w exponentiation algorithm and then to find an expansion of the exponent a of low weight that has coefficients only corresponding to the precomputed powers of g.

Example 11.1.25. Let $w = 4$. The standard sliding window exponentiation algorithm would require precomputing $g^3, g^5, g^7, g^9, g^{11}, g^{13}$ and g^{15}. Using w-NAFs requires precomputing g^3, g^5 and g^7 (as well as g^{-1}, g^{-3}, g^{-5} and g^{-7}, which is assumed to be easy). One could also consider a fractional w-NAF method whereby only g^3 and g^5 are computed.

Consider the exponent $a = 311$. This has the 4-NAF expansion

$$100030007$$

of weight 3. So one can compute g^{311} (ignoring the precomputation) using 8 squarings and 2 multiplications. This expansion cannot be used with the fractional 4-NAF, since we have not precomputed g^7. Instead, one finds the expansion

$$500\overline{1}00\overline{1},$$

which is not a 4-NAF (since each non-zero coefficient is not followed by 3 zero coefficients). However, it still has weight 3 and one computes g^{311} with 6 squarings and 2 multiplications (as well as faster precomputation).

Exercise 11.1.26. Compute a 4-NAF and a fractional 4-NAF as in Example 11.1.25 (i.e., again using only coefficients $\{0, \pm 1, \pm 3\}$) for $a = 887$.

An important issue for these methods is to determine the expected density of the resulting expansions. In Section 5.1 of Möller [432] the formula

$$\frac{1}{w + (1+m)/2^{w-1}}$$

(where the computed powers of g are $g^{\pm 1}, g^{\pm 3}, \ldots, g^{\pm m}$ and $w = \lfloor \log_2(m) \rfloor + 1$) for the density of a fractional window NAF is derived. This formula is verified using Markov chain methods in Theorem 1 of Schmidt-Samoa, Semay and Takagi [520]. Section 2.1 of [433] proves minimality of the weight among all expansions with that digit set.

All the above algorithms compute the signed expansion in a right-to-left manner. For some applications it may be desirable to compute expansions from left-to-right. There are a number of papers on this issue.

11.2 Multi-exponentiation

An n-dimensional **multi-exponentiation** (also called **simultaneous multiple exponentiation**) is the problem of computing a product $g_1^{a_1} \cdots g_n^{a_n}$. The question of how

[2] Sliding windows over NAFs were considered in Section IV.2.3 of [64]. Fractional sliding windows over NAFs seem to have been first used in [224].

efficiently this can be done was asked by Richard Bellman as problem 5125 of volume 70, number 6 of the American Mathematical Monthly in 1963. A solution was given by E. G. Straus[3] in [594]; the idea was re-discovered by Shamir and is often attributed to him. We only give a brief discussion of this topic and refer to Section 9.1.5 of [16] and Section 3.3.3 of [274] for further details.

Algorithm 7 computes an n-dimensional multi-exponentiation. We write $a_{i,j}$ for the j-th bit of a_i (where, as usual in this book, the least significant bit is $a_{i,0}$); if $j > \log_2(a_i)$ then $a_{i,j} = 0$. The main idea is to use a single accumulating variable (in this case called h) and to perform only one squaring. If a value g_i does not change between executions of the algorithm then we call it a **fixed base** and otherwise it is a **variable base** (the precomputation can be improved when some of the g_i are fixed). We assume that the integers a_i all vary.

Algorithm 7 Basic Multi-exponentiation

INPUT: $g_1, \ldots, g_n \in G, a_1, \ldots, a_n \in \mathbb{N}$
OUTPUT: $\prod_{i=1}^{n} g_i^{a_i}$
1: Precompute all $u_{b_1,\ldots,b_n} = \prod_{i=1}^{n} g_i^{b_i}$ for $b_i \in \{0, 1\}$
2: Set $l = \max_{1 \leq i \leq n} \{\lfloor \log_2(a_i) \rfloor\}$
3: $h = u_{a_{1,l},\ldots,a_{n,l}}$
4: $j = l - 1$
5: **while** $j \geq 0$ **do**
6: $\quad h = h^2$
7: $\quad h = h u_{a_{1,j},\ldots,a_{n,j}}$
8: $\quad j = j - 1$
9: **end while**
10: **return** h

Example 11.2.1. One can compute $g_1^7 g_2^5$ by setting $h = u_{1,1} = g_1 g_2$, then computing $h = h^2 = g_1^2 g_2^2$, $h = h u_{1,0} = g_1^3 g_2^2$, $h = h^2 = g_1^6 g_2^4$, $h = h u_{1,1} = g_1^7 g_2^5$.

Exercise 11.2.2. Show that one can perform the precomputation in Algorithm 7 in $2^n - n - 1$ multiplications. Show that the main loop of Algorithm 7 performs l squarings and l multiplications.

Exercise 11.2.3. (Yen, Laih, and Lenstra [638]) Show that by performing further precomputation one can obtain a sliding window multi-exponentiation algorithm that still requires l squarings in the main loop, but fewer multiplications. Determine the precomputation cost.

An alternative approach[4] to multi-exponentiation is called **interleaving**. The basic idea is to replace line 7 in Algorithm 7 by

$$\textbf{for } i = 1 \textbf{ to } n \textbf{ do } h = h g_i^{a_{i,j}} \textbf{ end for}$$

and to omit the precomputation. This version is usually less efficient than Algorithm 7 unless n is rather large. However the benefit of interleaving comes when using sliding windows: since the precomputation cost and storage requirements for the method in Exercise 11.2.3 are so high, it is often much more practical to use a sliding window version in the setting of interleaving. We refer to [431] and Section 3.3.3 of [274] for further discussion of this method.

[3] Straus had the remarkable ability to solve crossword puzzles in English (his third language) using only the horizontal clues; see his commemorative issue of the Pacific Journal of Mathematics.

[4] Independently discovered by Möller [431] and Gallant, Lambert and Vanstone [233].

Exercise 11.2.4. Write pseudocode for multi-exponentiation using interleaving and sliding windows.

Exercise 11.2.5. Write pseusocode for multi-exponentiation using interleaving and sliding windows over NAF expansions.

Another approach, when signed expansions are being used, is to find a representation for the exponents $a_1, \ldots, a_n$ so that the j-th component of the representations of all a_i is simultaneously zero relatively often. Such a method was developed by Solinas [577] and is called a **joint sparse form**. We refer to Section 9.1.5 of [16] and Section 3.3.3 of [274].

Multi-exponentiation for Algebraic Group Quotients

In algebraic group quotients, multiplication is not well-defined and so extra information is needed to be able to compute $\prod_{i=1}^{n} g_i^{a_i}$. A large survey of exponentiation algorithms and multi-exponentiation algorithms for algebraic group quotients is give in Chapter 3 of Stam's thesis [579]. In particular, he gives the Montgomery Euclidean ladder in Section 3.3 (also see Section 4.3 of [578]). Due to lack of space we do not discuss this topic further.

11.3 Efficient Exponentiation in Specific Algebraic Groups

We now discuss some exponentiation methods that exploit specific features of algebraic groups.

11.3.1 Alternative Basic Operations

So far, all exponentiation algorithms have been based on squaring (and hence have used representations of integers to the base 2). We now briefly mention some alternatives to squaring as the basic operation. First we discuss halving and tripling. Frobenius expansions will be discussed in Section 11.3.2.

When one has several possible basic operations then one can consider **multi-base representations** of integers for exponentiation. These ideas were first proposed by Dimitrov, Jullien and Miller [181] but we do not consider them further in this book.

Point Halving on Elliptic Curves

This idea, independently discovered by Knudsen [342] and Schroeppel, applies to subgroups of odd order in ordinary elliptic curves over finite fields of characteristic two. The formulae for point halving were given in Exercise 9.1.4: Given $P = (x_P, y_P) \in E(\mathbb{F}_{2^n})$ one finds $Q = (x_Q, y_Q) \in E(\mathbb{F}_{2^n})$ such that $[2]Q = P$ by solving $\lambda_Q^2 + \lambda_Q = x_P + a_2$. For either solution let $x_Q = \sqrt{x_P(\lambda_Q + 1) + y_P} = \sqrt{x_P(\lambda_P + \lambda_Q + x_P + 1)}$ and $y_Q = x_Q(\lambda_Q + x_Q)$. One must ensure that the resulting point Q has odd order. When $2 \| \#E(\mathbb{F}_{q^n})$ this is easy as, by Exercise 9.1.4, it is sufficient to check that $\mathrm{Tr}_{\mathbb{F}_{2^n}/\mathbb{F}_2}(x_Q) = \mathrm{Tr}_{\mathbb{F}_{2^n}/\mathbb{F}_2}(a_2)$. In practice it is more convenient to check whether $\mathrm{Tr}_{\mathbb{F}_{2^n}/\mathbb{F}_2}(x_Q^2) = \mathrm{Tr}_{\mathbb{F}_{2^n}/\mathbb{F}_2}(a_2)$.

Exercise 11.3.1. Write down the point halving algorithm.

Knudsen suggests representing points using the pair (x_P, λ_P) instead of (x_P, y_P). In any case, this can be done internally in the exponentiation algorithm. When $\mathbb{F}_{2^n}$ is represented using a normal basis over $\mathbb{F}_2$ then halving can be more efficient than doubling on such an elliptic curve. One can therefore use expansions of integers to the "base 2^{-1}" for efficient exponentiation. We refer to Section 13.3.5 of [16] and [342] for the details.

Tripling

Doche, Icart and Kohel [183] suggested to speed up the computation of $[m]P$ on E for small m by splitting it as $\hat{\phi} \circ \phi$ where $\phi : E \to E'$ is an isogeny of degree m. We refer to Exercise 9.6.30 for an example of this in the case $m = 3$, and to [183] for the details in general.

11.3.2 Frobenius Expansions

Koblitz (in Section 5 of [344]) presented a very efficient doubling formula for $E : y^2+y = x^3$ over $\mathbb{F}_2$ (see Exercise 9.1.3). Defining $\pi(x, y) = (x^2, y^2)$ one can write this as $[2]P = -\pi^2(P)$ for all $P \in E(\mathbb{F}_{2^m})$ for any integer m. We assume throughout this section that finite fields $\mathbb{F}_{p^m}$ are represented using a normal basis so that raising to the power p is very fast. Menezes and Vanstone [416] and Koblitz [346, 347] explored further how to speed up arithmetic on curves over small fields. However, the curves used in [344, 416, 346] are supersingular and so are less commonly used for cryptography.

The Frobenius map can be used to speed up elliptic curve exponentiation on more general curves. For cryptographic applications we assume that E is an elliptic curve over $\mathbb{F}_q$ such that $\#E(\mathbb{F}_{q^m})$ has a large prime divisor r for some $m > 1$.[5] Let π be the q-power Frobenius map on E. The trick is to replace an integer a with a sequence $a_0, \ldots, a_l$ of "small" integers such that

$$[a]P = \sum_{i=0}^{l} [a_i]\pi^i(P)$$

for the point $P \in E(\mathbb{F}_{q^m})$ of interest.

Definition 11.3.2. Let E be an elliptic curve over $\mathbb{F}_q$ and let π be the q-power Frobenius map. Let $S \subset \mathbb{Z}$ be a finite set such that $0 \in S$ (the set S is usually obvious from the context). A **Frobenius expansion** with **digit set** S is an endomorphism of the form

$$\sum_{i=0}^{l} [a_i]\pi^i$$

where $a_i \in S$ and $a_l \neq 0$. The **length** of a Frobenius expansion is $l + 1$. The **weight** of a Frobenius expansion is the number of non-zero a_i.

Many papers write τ for the Frobenius map and speak of τ-adic expansions, but we will call them **π-adic expansions** in this book.

Example 11.3.3. Let $E : y^2 + xy = x^3 + ax^2 + 1$ over $\mathbb{F}_2$ where $a \in \{0, 1\}$. Consider the group $E(\mathbb{F}_{2^m})$ and write $\pi(x, y) = (x^2, y^2)$. From Exercise 9.10.11 we know that $\pi^2 + (-1)^a\pi + 2 = 0$. Hence, one can replace the computation $[2]P$ by $-\pi(\pi(P)) - (-1)^a\pi(P)$. At first sight, there is no improvement here (we have replaced a doubling with an elliptic curve addition). However, the idea is to represent an integer by a polynomial in π. For example, one can verify that

$$-T^5 + T^3 + 1 \equiv 9 \pmod{T^2 + T + 2}$$

and so one can compute $[9]P$ (normally taking 3 doublings and an addition) as $-\pi^5(P) + \pi^3(P) + P$ using only two elliptic curve additions.

[5] Note that, for any fixed elliptic curve E over $\mathbb{F}_q$ and any fixed $c \in \mathbb{R}_{>0}$, it is not known if there are infinitely many $m \in \mathbb{N}$ such that $\#E(\mathbb{F}_{q^m})$ has a prime factor r such that $r > cq^m$. However, in practice one finds a sufficient quantity of suitable examples.

This idea can be extended to any algebraic group G (in particular, an elliptic curve, the divisor class group of a hyperelliptic curve, or an algebraic torus) that is defined over a field $\mathbb{F}_q$ but for which one works in the group $G(\mathbb{F}_{q^m})$.

Exercise 11.3.4. Give an algorithm to compute $[a]P$ when $a = \sum_{i=0}^{l} a_i \pi^i(P)$ is a Frobenius expansion. What is the cost of the algorithm?

Definition 11.3.5. Let $S \subseteq \mathbb{Z}$ such that $0 \in S$ and if $a \in S$ then $-a \in S$. Let $a = \sum_{i=0}^{l} a_i \pi^i$ be a Frobenius expansion with $a_i \in S$. Then a is in **non-adjacent form** if $a_i a_{i+1} = 0$ for all $0 \leq i < l$. Such an expansion is also called a π**-NAF**.

An important task is to convert an integer n into a Frobenius expansion in non-adjacent form. In fact, to get short expansions we will need to convert a general element $n_0 + n_1\pi$ to a π-NAF, so we study the more general problem. The crucial result is the following.

Lemma 11.3.6. *Let π satisfy $\pi^2 - t\pi + q = 0$. An element $n_0 + n_1\pi$ in $\mathbb{Z}[\pi]$ is divisible by π if and only if $q \mid n_0$. In this case*

$$(n_0 + n_1\pi)/\pi = (n_1 + tn_0/q) + \pi(-n_0/q). \tag{11.1}$$

Similarly, it is divisible by π^2 if and only if $q \mid n_0$ and $qn_1 \equiv -tn_0 \pmod{q^2}$.

Proof: Note that $\pi^2 = t\pi - q$. Since

$$\pi(m_0 + m_1\pi) = -qm_1 + \pi(m_0 + tm_1)$$

it follows that $n_0 + n_1\pi = \pi(m_0 + m_1\pi)$ if and only if

$$n_0 = -qm_1, \quad \text{and} \quad n_1 = m_0 + tm_1.$$

Writing $m_1 = -n_0/q$ and $m_0 = n_1 - tm_1$ yields equation (11.1).

Repeating the argument, one can divide the element in equation (11.1) by π if and only if $q \mid (n_1 + tn_0/q)$. The result follows. □

The idea of the algorithm for computing a π-NAF, given $n_0 + n_1\pi$, is to add a suitable integer so that the result is divisible by π^2, divide by π^2, then repeat. This approach is only really practical when $q = 2$, so we restrict to this case.

Lemma 11.3.7. *Let $\pi^2 - t\pi + 2 = 0$ where $t = \pm 1$. Then $n_0 + n_1\pi$ is either divisible by π or else there is some $\epsilon = \pm 1$ such that*

$$(n_0 + \epsilon) + n_1\pi \equiv 0 \pmod{\pi^2}.$$

Indeed, $\epsilon = (n_0 + 2n_1 \pmod 4) - 2$, if one defines $n_0 + 2n_1 \pmod 4 \in \{1, 3\}$.

Proof: If $\pi \nmid (n_0 + n_1\pi)$ then n_0 is odd and so $n_0 \pm 1$ is even. One can choose the sign such that $2n_1 \equiv -(n_0 \pm 1) \pmod 4$ in which case the result follows. □

The right-to-left algorithm[6] to generate a π-NAF is then immediate (see Algorithm 8; this algorithm computes $u = -\epsilon$ in the notation of Lemma 11.3.7). To show that the algorithm terminates we introduce the norm map: for any $a, b \in \mathbb{R}$ define $\mathrm{N}(a + b\pi) = (a+b\pi)(a+b\bar{\pi}) = a^2+tab+qb^2$ where $\pi, \bar{\pi} \in \mathbb{C}$ are the roots of the polynomial $x^2-tx+q = 0$. This map agrees with the norm map with respect to the quadratic field extension $\mathbb{Q}(\pi)/\mathbb{Q}$ and so is multiplicative. Note also that $\mathrm{N}(a+b\pi) \geq 0$ and equals zero only when

[6] Solinas states that this algorithm was joint work with R. Reiter.

$a = b = 0$. Meier and Staffelbach [415] note that, if $n_0 + n_1\pi$ is divisible by π, then $\mathrm{N}((n_0 + n_1\pi)/\pi) = \frac{1}{2}\mathrm{N}(n_0 + n_1\pi)$. This suggests that the length of the π-NAF will grow like $\log_2(\mathrm{N}(n_0 + n_1\pi))$. The case $\mathrm{N}((n_0 \pm 1 + n_1\pi)/\pi)$ needs more care. Lemma 3 of Meier and Staffelbach [415] states that if $\mathrm{N}(n_0 + n_1\pi) < 2^n$ then there is a corresponding Frobenius expansion[7] of length at most n. Theorem 2 of Solinas gives a formula for the norm in terms of the length $k = l+1$ of the corresponding π-NAF, from which he deduces (equation (53) of [576])

$$\log_2(\mathrm{N}(n_0 + n_1\pi)) - 0.55 < k < \log_2(\mathrm{N}(n_0 + n_1\pi)) + 3.52$$

when $k \geq 30$.

Algorithm 8 Convert $n_0 + n_1\pi$ to non-adjacent form

INPUT: $n_0, n_1 \in \mathbb{Z}$
OUTPUT: $a_0, \ldots, a_l \in \{-1, 0, 1\}$

```
1: while n_0 ≠ 0 and n_1 ≠ 0 do
2:     if n_0 odd then
3:         u = 2 − (n_0 + 2n_1 (mod 4))
4:         n_0 = n_0 − u
5:     else
6:         u = 0
7:     end if
8:     Output u
9:     (n_0, n_1) = (n_1 + t n_0/2, −n_0/2)
10: end while
```

Example 11.3.8. Suppose $\pi^2 + \pi + 2 = 0$. To convert $-1 + \pi$ to a π-NAF one writes $n_0 = -1$ and $n_1 = 1$. Let $u = 2 - (n_0 + 2n_1 \pmod 4) = 2 - (1) = 1$. Output 1 and set $n_0 = n_0 - 1$ to get $-2 + \pi$. Dividing by π yields $2 + \pi$. One can divide by π again (output 0 first) to get $-\pi$, output 0 and divide by π again to get -1. The π-NAF is therefore $1 - \pi^3$.

To see this directly using the equation $\pi^2 + \pi + 2 = 0$ write

$$-1 + \pi + (\pi^2 + \pi + 2)(1 - \pi) = 1 - \pi^3.$$

Exercise 11.3.9. Verify that Algorithm 8 does output a π-NAF.

Exercise 11.3.10. Let $\pi^2 - \pi + 2 = 0$. Use Algorithm 8 to convert $107 + 126\pi$ into non-adjacent form.

Exercise 11.3.11. Show, using the same methods as Lemma 11.1.14, that the average density of a π-NAF tends to $1/3$ when $q = 2$.

Reducing the Length of Frobenius Expansions

As we have seen, $\mathrm{N}(n + 0\pi) = n^2$, while the norm only decreases by a factor of roughly 2 each time we divide by π. Hence, the Frobenius expansions output by Algorithm 8 on input n have length roughly $2\log_2(n)$. Since the density is $1/3$ it follows that the weight of the Frobenius expansions is roughly $\frac{2}{3}\log_2(n)$. Exponentiation using Frobenius expansions is therefore faster than using the square-and-multiply algorithm, even with

[7] Meier and Staffelbach do not consider Frobenius expansions in non-adjacent form.

sliding windows (since the latter method always needs $\log_2(n)$ doublings and also some additions).

However, it is a pity that the expansions are so long. It is natural to seek shorter expansions that still have the same density. The crucial observation, due to Meier and Staffelbach, is that Algorithm 8 outputs a Frobenius expansion $\sum_i [a_i]\pi^i$ that acts the same as $[n]$ on all points in $E(\overline{\mathbb{F}}_q)$ whereas, for a given application, one only needs a Frobenius expansion that acts the same as $[n]$ on the specific subgroup $\langle P \rangle$ of prime order r.

Definition 11.3.12. Let E be an elliptic curve over $\mathbb{F}_p$, $P \in E(\mathbb{F}_{p^m})$, and let π be the p-power Frobenius map on E. We say that two Frobenius expansions $a(\pi), b(\pi) \in \mathbb{Z}[\pi]$ are **equivalent** with respect to P if

$$a(\pi)(P) = b(\pi)(P).$$

Exercise 11.3.13. Let the notation be as in Definition 11.3.12. Show that if $a(\pi) \equiv b(\pi) \pmod{\pi^m - 1}$ then $a(\pi)$ and $b(\pi)$ are equivalent with respect to P.

Show that if $Q \in \langle P \rangle$ and $a(\pi)$ and $b(\pi)$ are equivalent with respect to P then $a(\pi)$ and $b(\pi)$ are equivalent with respect to Q.

A simple idea is to replace all powers π^i by $\pi^{i \pmod m}$. This will reduce the length of a Frobenius expansion but it does not significantly change the weight (and hence, the cost of exponentiation does not change).

The goal is therefore to find an element $n_0 + n_1\pi$ of "small" norm that is equivalent to $[n]$ with respect to P. Then one applies Algorithm 8 to the pair (n_0, n_1), not to $(n, 0)$. There are two simple ways to find an element of small norm, both of which apply the Babai rounding method (see Section 18.2) in a suitable lattice. They differ in how one expresses the fact that $(n_0 + n_1\pi)P = [n]P$ for the point P of interest.

- Division with remainder in $\mathbb{Z}[\pi]$.

  This method was proposed by Meier and Staffelbach [415] and is also used by Solinas (Section 5.1 of [576]). Since $(\pi^m - 1)(P) = \mathcal{O}_E$ when $P \in E(\mathbb{F}_{q^m})$ one wants to determine the remainder of dividing n by $(\pi^m - 1)$. The method is to consider the element $\gamma = n/(\pi^m - 1) \in \mathbb{R}[\pi]/(\pi^2 - t\pi + q)$ and find a close vector to it (using Babai rounding) in the lattice $\mathbb{Z}[\pi]$. In other words, write $\gamma = \gamma_0 + \gamma_1\pi$ with $\gamma_0, \gamma_1 \in \mathbb{R}$ and round them to the nearest integers g_0, g_1 (in the special case of $\pi^2 \pm \pi + 2 = 0$ there is an exact description of a fundamental domain for the lattice that can be used to "correct" the Babai rounding method if it does not reach the closest lattice element). Lemma 3 of [415] and Proposition 57 of [576] state that $\mathrm{N}(\gamma - (g_0 + g_1\pi)) \leq 4/7$. One can then define

  $$n_0 + n_1\pi = n - (g_0 + g_1\pi)(\pi^m - 1) \pmod{\pi^2 - t\pi + q}.$$

- The Gallant-Lambert-Vanstone method [233].

  This method appears in a different context (see Section 11.3.3), but it is also suitable for the present application. We assume that $P \in E(\mathbb{F}_{q^m})$ has prime order r where $r \| \#E(\mathbb{F}_{q^m})$. Since $\pi(P) \in E(\mathbb{F}_{q^m})$ has order r it follows that $\pi(P) = [\lambda]P$ for some $\lambda \in \mathbb{Z}/r\mathbb{Z}$. The problem is therefore to find small integers n_0 and n_1 such that

  $$n_0 + n_1\lambda \equiv n \pmod{r}.$$

One defines the **GLV lattice**

$$L = \{(x_0, x_1) \in \mathbb{Z}^2 : x_0 + x_1\lambda \equiv 0 \pmod{r}\}.$$

A basis for L is given in Exercise 11.3.22. The idea is to find a lattice vector $(n_0', n_1') \in L$ close to $(n, 0)$. Then $|n_1'|$ is "small" and $|n - n_0'|$ is "small". Define $n_0 = n - n_0'$ and $n_1 = -n_1'$ so that

$$n_0 + n_1\lambda \equiv n \pmod{r}$$

as required.

We can compute a reduced basis for the lattice and then use Babai rounding to solve the closest vector problem (CVP). Note that the reduced basis can be precomputed. Since the dimension is two, one can use the Lagrange-Gauss lattice reduction algorithm (see Section 17.1). Alternatively, one can use Euclid's algorithm to compute (n_0, n_1) directly (as discussed in Section 17.1.1, Euclid's algorithm is closely related to the Lagrange-Gauss algorithm).

Example 11.3.14. The elliptic curve $E : y^2 + xy = x^3 + x^2 + 1$ over $\mathbb{F}_{2^{19}}$ has $2r$ points where $r = 262543$ is prime. Let $\pi(x, y) = (x^2, y^2)$. Then $\pi^2 - \pi + 2 = 0$. Let $n = 123456$. We want to write n as $n_0 + n_1\pi$ on the subgroup of $E(\mathbb{F}_{2^{19}})$ of order r.

For the "division with remainder in $\mathbb{Z}[\pi]$" method we first use Lucas sequences (as in Exercise 9.10.10) to determine that

$$\pi^{19} - 1 = -(171 + 457\pi)$$

(one can think of this as equality of complex numbers where π is a root of $x^2 - x + 2$, or as congruence of polynomials modulo $\pi^2 - \pi + 2$). It is convenient to change the sign (the method works in both cases). The norm of $171 + 457\pi$ is $\#E(\mathbb{F}_{2^{19}}) = 2r = 525086$. and so

$$\frac{n}{-\pi^{19} + 1} = \frac{n(171 + 457\bar{\pi})}{525086} \approx 147.653 - 107.448\pi.$$

(since $\bar{\pi} = 1 - \pi$). Rounding gives $148 - 107\pi$ and

$$n - (148 - 107\pi)(171 + 457\pi) \equiv 350 - 440\pi \pmod{\pi^2 - \pi + 2}.$$

This is a short representative for n, but its norm is larger than $8r/7$, which is not optimal. Section 5.1 of Solinas [576] shows how to choose a related element of smaller norm. In this case the correct choice of rounding is $147 - 107\pi$ giving

$$n - (147 - 107\pi)(171 + 457\pi) \equiv 521 + 17\pi \pmod{\pi^2 - \pi + 2},$$

which has norm less than $8r/7$.

Now for the Gallant-Lambert-Vanstone method. We compute $\gcd(x^{19} - 1, x^2 - x + 2) = (x - \lambda)$ in $\mathbb{F}_r[x]$, where $\lambda = 84450$. The lattice with (row) basis

$$\begin{pmatrix} r & 0 \\ -\lambda & 1 \end{pmatrix}$$

has LLL (or Lagrange-Gauss) reduced basis

$$B = \begin{pmatrix} 171 & 457 \\ 457 & -314 \end{pmatrix}.$$

Writing $\underline{b}_1, \underline{b}_2$ for the rows of the reduced matrix one finds $(n, 0) \approx 147.65\underline{b}_1 + 214.90\underline{b}_2$. One computes

$$(n, 0) - (148, 215)\, B = (-107, -126)\,.$$

One can verify that $-107 - 126\lambda \equiv n \pmod{r}$. (Exercise 11.3.16 shows how to get this element using remainders in $\mathbb{Z}[\pi]$.) The corresponding Frobenius expansion can be obtained from the solution to Exercise 11.3.10.

Exercise 11.3.15. Prove that both the above methods yield an element $n_0 + n_1\pi \in \mathbb{Z}[\pi]$ that is equivalent to n.

Exercise 11.3.16. Show that if $P \in E(\mathbb{F}_{q^m})$ but $P \notin E(\mathbb{F}_q)$ then instead of computing the remainder in $\mathbb{Z}[\pi]$ modulo the polynomial $(\pi^m - 1)$ one can use $(\pi^m - 1)/(\pi - 1)$. Repeat Example 11.3.14 using this polynomial.

In practice it is unnecessary to determine the minimal solution (n_0, n_1) as long as n_0 and n_1 have bit-length roughly $\frac{1}{2}\log_2(r)$ (where the point P has order r). We also stress that computing the q-power Frobenius map π is assumed to be very fast, so the main task is to minimise the *weight* of the representation, not its length.

Remark 11.3.17. In cryptographic protocols one is often computing $[a]P$, where a is a randomly chosen integer modulo r. Rather than choosing a random integer a and then converting to a Frobenius expansion, one could choose a random Frobenius expansion of given weight and length (this trick appears in Section 6 of [347] where it is attributed to H. W. Lenstra Jr.).

We have analysed π-NAFs in the case $q = 2$. Müller [443] gives an algorithm to compute Frobenius expansions for elliptic curves over $\mathbb{F}_{2^e}$ with $e > 1$ (but still small). The coefficients of the expansion lie in $\{-2^{e-1}, \ldots, 2^{e-1}\}$. Smart [571] gives an algorithm for odd q, with a similar coefficient set; see Exercise 11.3.18. Lange [368] generalises to hyperelliptic curves. In all cases, the output is not necessarily in non-adjacent form; to obtain a π-NAF in these cases seems to require much larger digit sets. In any case, the asymptotic density of π-NAFs with large digit set is not significantly smaller than $1/2$ and this can easily be bettered using window methods (see Exercise 11.3.19).

Exercise 11.3.18. Let $q > 2$. Show that Algorithm 8 can be generalised (not to compute a π-NAF, but just a π-adic expansion) by taking digit set $\{-\lfloor q/2 \rfloor, \ldots, -1, 0, 1, \ldots, \lfloor q/2 \rfloor\}$ (or this set with $-\lfloor q/2 \rfloor$ removed when $q > 2$ is even).

Exercise 11.3.19. Let E be an elliptic curve over a field $\mathbb{F}_q$, let π be the q-power Frobenius map, and let $P \in E(\mathbb{F}_{q^m})$. Let $S = \{-(q-1)/2, \ldots, -1, 0, 1 \ldots, (q-1)/2\}$ if q is odd and $S = \{-(q-2)/2, \ldots, -1, 0, 1, \ldots, q/2\}$ if q is even.

Suppose one has a Frobenius expansion

$$a(\pi) = \sum_{j=0}^{l} [a_j]\pi^j$$

with $a_j \in S$. Let $w \in \mathbb{N}$. Give a sliding window method to compute $[a(\pi)]P$ using windows of length w. Give an upper bound on the cost of this algorithm (including pre-computation) ignoring the cost of evaluating π.

Exercise 11.3.20. (Brumley and Järvinen [112]) Let E be an elliptic curve over $\mathbb{F}_q$, π be the q-power Frobenius, and $P \in E(\mathbb{F}_{q^m})$ have prime order r where $r \| \#E(\mathbb{F}_{q^m})$. Given a Frobenius expansion $a(\pi) = \sum_i [a_i]\pi^i$ show how to efficiently compute $a \in \mathbb{Z}$ such that $a(\pi)(P) = [a]P$.

Dimitrov, Järvinen, Jacobson, Chan and Huang [180] use Frobenius expansions on Koblitz curves to obtain a method for computing $[k]P$ which is provably sub-linear (i.e., using $o(\log(k))$ field operations). For a complete presentation of Frobenius expansions, and further references, we refer to Section 15.1 of [16]. For multi-exponentiation using Frobenius expansions there is also a π-adic joint sparse form; see Section 15.1.1.e of [16] for details.

11.3.3 GLV Method

This method is due to Gallant, Lambert and Vanstone [233] for elliptic curves and Stam and Lenstra (see Section 4.4 of [580]) for tori.[8] The idea is to use an "efficiently computable" (see below for a clarification of this term) group homomorphism ψ and replace the computation g^a in a group of order r by the multi-exponentiation $g^{a_0}\psi(g)^{a_1}$ for suitable integers a_0 and a_1 such that $|a_0|, |a_1| \approx \sqrt{r}$. Typical choices for ψ are an automorphism of an elliptic curve or the Frobenius map on an elliptic curve or torus over an extension field.

More precisely, let $g \in G(\mathbb{F}_q)$ be an element of prime order r in an algebraic group and let ψ be a group homomorphism such that $\psi(g) \in \langle g \rangle$ (this is automatic if $\psi : G(\mathbb{F}_q) \to G(\mathbb{F}_q)$ and $r \| \#G(\mathbb{F}_q)$). Then $\psi(g) = g^\lambda$ for some $\lambda \in \mathbb{Z}/r\mathbb{Z}$. The meaning of "efficiently computable" is essentially that computing $\psi(g)$ is much faster than computing g^λ using exponentiation algorithms. Hence, we require that λ and $r - \lambda$ are not small; in particular, the map $\psi(P) = -P$ on an elliptic curve is not interesting for this application.

Example 11.3.21. Consider $\mathbb{T}_2(\mathbb{F}_{p^2})$, with elements represented as in Definition 6.3.7 so that $u \in \mathbb{F}_{p^2}$ corresponds to $g = (u+\theta)/(u+\overline{\theta}) \in \mathbb{F}_{p^4}$. It follows by Lemma 6.3.12 that $A - u^p$ (where $\theta^2 + A\theta + B = 0$) corresponds to $g^p = (u^p + \overline{\theta})/(u^p + \theta) = \big((u^p + \theta)(u^p + \overline{\theta})\big)^{-1}$. Since computing u^p for $u \in \mathbb{F}_{p^2}$ is easy, the map $\psi(u) = A - u^p$ is a useful efficiently computable group homomorphism with respect to the torus group operation.

One can also perform exponentiation in $G_{q^2,2} \subseteq \mathbb{F}_{p^4}^*$ using Frobenius. Given an exponent a such that $1 \leq a < p^2$ one lets a_0 and a_1 be the coefficients in the base-p representation of a and computes g^a as $g^{a_0}\,(g^p)^{a_1}$. Note that g^p is efficient to compute as it is a linear map on the 4-dimensional vector space $\mathbb{F}_{p^4}$ over $\mathbb{F}_p$.

Other examples include the automorphism ζ_3 on $y^2 = x^3 + B$ in Example 9.4.2 and the automorphisms in Exercises 9.4.5 and 10.2.12. Computing the eigenvalue λ for ψ is usually easy in practice: for elliptic curves λ is a root of the characteristic polynomial of ψ modulo r.

In some applications (for example, $\mathbb{T}_2(\mathbb{F}_{p^3})$ or some automorphisms on genus 2 curves such as the one in Exercise 10.2.12) one can replace g^a by $g^{a_0}\psi(g)^{a_1} \cdots \psi^{l-1}(g)^{a_{l-1}}$ for some $l > 2$. We call this the l-dimensional GLV method. We stress that l cannot be chosen arbitrarily; in Example 11.3.21 the map ψ^2 is the identity map and so is not useful.

In the previous section we sketched, for elliptic curves, the GLV method to represent an integer as a short Frobenius expansion with relatively small coefficients. One can do the same for any endomorphism ψ as long as $\psi(P) = [\lambda]P$ (or $\psi(g) = g^\lambda$ in multiplicative notation). The **GLV lattice** is

$$L = \{(x_0, \ldots, x_l) \in \mathbb{Z}^{l+1} : x_0 + x_1\lambda + \cdots + x_l\lambda^l \equiv 0 \pmod{r}\}.$$

[8] A patent on the method was filed by Gallant, Lambert and Vanstone in 1999.

A basis for L is given in Exercise 11.3.22. As explained earlier, to convert an integer a into GLV representation one finds a lattice vector $(a'_0, a'_1, \ldots, a'_l) \in L$ close to $(a, 0, \ldots, 0)$ (using Babai rounding) then sets $a_0 = a - a'_0$ and $a_i = -a'_i$ for $1 \leq i \leq l$.

Exercise 11.3.22. Show that

$$\begin{pmatrix} r & 0 & 0 & \cdots & 0 \\ -\lambda & 1 & 0 & \cdots & 0 \\ -\lambda^2 & 0 & 1 & \cdots & 0 \\ \vdots & & & & \vdots \\ -\lambda^l & 0 & 0 & \cdots & 1 \end{pmatrix}$$

is a basis for the GLV lattice L.

Exercise 11.3.23. Show how to compute the coefficients $a_0, \ldots, a_l$ for the GLV method using Babai rounding.

Exercise 11.3.24 gives a construction of homomorphisms for the GLV method that apply to a large class of curves. We refer to Galbraith, Lin and Scott [224] for implementation results that show the benefit of using this construction.

Exercise 11.3.24. (Iijima, Matsuo, Chao and Tsujii [305]) Let $p > 3$ be a prime and let $E : y^2 = x^3 + a_4x + a_6$ be an ordinary elliptic curve over $\mathbb{F}_p$ with $p + 1 - t$ points (note that $t \neq 0$). Let $u \in \mathbb{F}_{p^2}^*$ be a non-square and define $E' : Y^2 = X^3 + u^2a_4X + u^3a_6$ over $\mathbb{F}_{p^2}$. Show that E' is the quadratic twist of $E(\mathbb{F}_{p^2})$ and that $\#E'(\mathbb{F}_{p^2}) = (p-1)^2 + t^2$. Let $\phi : E \to E'$ be the isomorphism $\phi(x, y) = (ux, u^{3/2}y)$ defined over $\mathbb{F}_{p^4}$.

Let $\pi(x, y) = (x^p, y^p)$ and define

$$\psi = \phi \circ \pi \circ \phi^{-1}.$$

Show that $\psi : E' \to E'$ is an endomorphism of E' that is defined over $\mathbb{F}_{p^2}$. Show that $\psi^2 = [-1]$.

Let $r \mid \#E'(\mathbb{F}_{p^2})$ be a prime such that $r > 2p$ and $r^2 \nmid \#E'(\mathbb{F}_{p^2})$. Let $P \in E'(\mathbb{F}_{p^2})$ have order r. Show that $\psi^2(P) - [t]\psi(P) + [p]P = \mathcal{O}_{E'}$. Hence deduce that $\psi(P) = [\lambda]P$ where $\lambda = t^{-1}(p-1) \pmod{r}$. Note that it is possible for $\#E'(\mathbb{F}_{p^2})$ to be prime, since E' is not defined over $\mathbb{F}_p$.

As in Remark 11.3.17, for some applications one might be able to choose a random GLV expansion directly, rather than choosing a random integer and converting it to GLV representation.

There is a large literature on the GLV method, including several different algorithms to compute the integers $a_0, \ldots, a_l$. As noted earlier, reducing the bit-length of the a_i by one or two bits makes very little effect on the overall running time. Instead, the weight of the entries $a_0, \ldots, a_l$ is more critical. We refer to Sections 15.2.1 and 15.2.2 of [16] for further details and examples of the GLV method. Section 15.2.3 of [16] discusses combining the GLV method with Frobenius expansions.

11.4 Sampling from Algebraic Groups

A natural problem, given an algebraic group G over a finite field $\mathbb{F}_q$, is to generate a "random" element of G. By "random" we usually mean uniformly at random from $G(\mathbb{F}_q)$ although sometimes it may be appropriate to weaken this condition. The first problem is to generate a random integer in $[0, p-1]$ or $[1, p-1]$. Examples 11.4.1 and 11.4.2 give two simple approaches. Chapter 7 of Sidorenko [562] is a convenient survey.

Example 11.4.1. One way to generate a random integer in $[0, p-1]$ is to generate random binary strings x of length k (where $2^{k-1} < p \leq 2^k$) and only output those satisfying $0 \leq x \leq p-1$.

Example 11.4.2. Another method is to generate a binary string that is longer than p and then return this value reduced modulo p. We refer to Section 7.4 of Shoup [556] for a detailed analysis of this method (briefly, if $2^{k-1} < p \leq 2^k$ and one generates a $k+l$ bit string then the statistical difference of the output from uniform is $1/2^l$). Section 7.5 of [556] discusses how to generate a random k-bit prime and Section 7.7 of [556] discusses how to generate a random integer of known factorisation.

Exercise 11.4.3. Show that the expected number of trials of the algorithm in Example 11.4.1 is less than 2.

Exercise 11.4.4. Give an algorithm to generate an element of $\mathbb{F}_{p^n}^*$ uniformly at random, assuming that generating random integers modulo p is easy.

Appendix B.2.4 of Katz and Lindell [334] gives a thorough discussion of sampling randomly in $(\mathbb{Z}/N\mathbb{Z})^*$ and $\mathbb{F}_p^*$.

Algorithm 5 shows how to compute a generator for $\mathbb{F}_p^*$, when the factorisation of $p-1$ is known. Generalising this algorithm to $\mathbb{F}_{p^n}^*$, when the factorisation of $p^n - 1$ is known, is straightforward. In practice one often works in a subgroup $G \subseteq \mathbb{F}_q^*$ of prime order r. To sample uniformly from G one can generate a uniform element in $\mathbb{F}_q^*$ and then raise to the power $(q-1)/r$. This exponentiation can be accelerated using any of the techniques discussed earlier in this chapter.

Exercise 11.4.5. Let q be a prime power such that the factorisation of $q-1$ is known. Give an algorithm to determine the order of an element $g \in \mathbb{F}_q^*$.

Exercise 11.4.6. Let q be a prime power. Let G be a subgroup of $\mathbb{F}_q^*$ such that the factorisation of the order of G is known. Give an algorithm to compute a generator of G.

An alternative approach to the sampling problem for a finite Abelian group G is given in Exercise 11.4.7 (these ideas will also be used in Exercise 15.5.2). However, this method is often not secure for applications in discrete logarithm cryptography. The reason is that one usually wants to sample group elements at random such that no information about their discrete logarithm is known, whereas the construction in Exercise 11.4.7 (especially when used to define a hash function) may give an attacker a way to break the cryptosystem.

Exercise 11.4.7. Let G be a finite Abelian group. Let $g_1, \ldots, g_k$ be fixed elements that generate G. Let $m_1, \ldots, m_k$ be the orders of $g_1, \ldots, g_k$ respectively. Then one can generate an element of G at random by choosing integers $a_1, \ldots, a_k$ uniformly at random such that $0 \leq a_i < m_i$ for $1 \leq i \leq k$ and computing

$$\prod_{i=1}^{k} g_i^{a_i}. \tag{11.2}$$

Show that this process does sample from G with uniform distribution.

11.4.1 Sampling from Tori

We now mention further techniques to speed up sampling from subgroups of $\mathbb{F}_q^*$.

Example 11.4.8. Lemma 6.3.4 shows that $\mathbb{T}_2(\mathbb{F}_q)$ and $G_{2,q} \subseteq \mathbb{F}_{q^2}^*$ are in one-to-one corrrespondence with the set

$$\{1\} \cup \{(a+\theta)/(a+\overline{\theta}) : a \in \mathbb{F}_q\}.$$

Hence, one can sample from $\mathbb{T}_2(\mathbb{F}_q)$ or $G_{2,q}$ as follows: Choose uniformly $0 \leq a \leq p$ and, if $a = p$, output 1, otherwise output $(a+\theta)/(a+\overline{\theta})$.

Generating elements of $\mathbb{T}_6(\mathbb{F}_q)$ or $G_{6,q}$ uniformly at random is less simple, since the compression map does not map to the whole of $\mathbb{A}^2(\mathbb{F}_q)$. Indeed, the group $G_{6,q}$ has $q^2-q+1 < q^2$ elements. Example 6.4.4 showed, in the case $q \equiv 2, 5 \pmod 9$, how to map

$$A = \{(a,b) \in \mathbb{A}^2(\mathbb{F}_q) : a^2 - ab + b^2 \neq 1\} \tag{11.3}$$

to a subset of $\mathbb{T}_6(\mathbb{F}_q)$.

Exercise 11.4.9. Let $q \equiv 2, 5 \pmod 9$. Give an algorithm to generate points in the set A of equation (11.3) uniformly at random. Hence, show how to efficiently choose random elements of a large subset of $\mathbb{T}_6(\mathbb{F}_q)$ or $G_{6,q}$ uniformly at random.

11.4.2 Sampling from Elliptic Curves

Let $E : y^2 = x^3 + a_4x + a_6$ be an elliptic curve over $\mathbb{F}_q$ where q is not a power of 2. To generate points in $E(\mathbb{F}_q)$ one can proceed as follows: choose a random $x \in \mathbb{F}_q$; test whether $x^3+a_4x+a_6$ is a square in $\mathbb{F}_q$; if not then repeat, otherwise take square roots to get y and output (uniformly) one of $\pm y$. It is not surprising that an algorithm to generate random points is randomised, but something that did not arise previously is that this algorithm uses a randomised subroutine (i.e., to compute square roots efficiently) and may need to be repeated several times before it succeeds (i.e., it is a Las Vegas algorithm). Hence, only an expected run time for the algorithm can be determined.

A more serious problem is that the output is not uniform. For example, $\mathcal{O}_E$ is never output, and points $(x, 0)$ occur with probability twice the probability of (x, y) with $y \neq 0$. A solution for all elliptic curves (and also hyperelliptic curves with imaginary model) is given in Algorithm 9. For a detailed analysis and generalisation of this algorithm see von zur Gathen, Shparlinski and Sinclair [240].

Exercise 11.4.10. Determine expected number of iterations of Algorithm 9 in the case of elliptic curves and hence the expected running time.

Deterministic Sampling of Elliptic Curve Points

The above methods are randomised, not just due to the randomness that naturally arises when sampling, but also because of the use of randomised algorithms for solving quadratic equations, and because not every x in the field is an x-cordinate of an elliptic curve point. It is of interest to minimise the reliance on randomness, especially when using the above ideas to construct a hash function (otherwise, there may be timing attacks). We first give an easy example.

Exercise 11.4.11. (Boneh and Franklin [80]) Let $p \equiv 2 \pmod 3$ be prime. Consider the supersingular elliptic curve $E : y^2 = x^3 + a_6$ over $\mathbb{F}_p$. One can sample points uniformly in $E(\mathbb{F}_p) - \{\mathcal{O}_E\}$ by uniformly choosing $y \in \mathbb{F}_p$ and setting $x = (y^2 - a_6)^{1/3} \pmod p$. The cube root is computed efficiently by exponentiation to the power $(2p-1)/3 \equiv 3^{-1} \pmod{p-1}$.

Algorithm 9 Near-uniform sampling of points on curves

INPUT: $H(x), F(x) \in \mathbb{F}_q[x]$ such that $C : y^2 + H(x)y = F(x)$ has one $\mathbb{F}_q$ point at infinity
OUTPUT: $P \in C(\mathbb{F}_q)$

1: Choose uniformly $0 \leq x_0 \leq q$
2: **if** $x_0 = q$ **then**
3: $S = \{$ point at infinity $\}$
4: **else**
5: Compute $S = \{(x_0, y_0) : y_0 \in \mathbb{F}_q \text{ and } y_0^2 + H(x_0)y_0 - F(x_0) = 0\}$
6: **end if**
7: **if** $\#S = 0$ **then**
8: goto line 1
9: **end if**
10: **if** $\#S = 1$ **then**
11: Choose uniformly $b \in \{0, 1\}$
12: **if** $b = 0$ **then**
13: Let $P \in S$
14: **else**
15: goto line 1
16: **end if**
17: **end if**
18: **if** $\#S = 2$ **then**
19: Let P be chosen randomly from S
20: **end if**
21: **return** P

The first general results on deterministic methods to find points on curves over finite fields $\Bbbk$ are due to Schinzel and Skałba [514]. Given $a_6 \in \Bbbk$ (the case $\mathrm{char}(\Bbbk) = 2$ is not interesting since the curve is singular, and the case $\mathrm{char}(\Bbbk) = 3$ is easy since taking cube roots is easy, so assume $\mathrm{char}(\Bbbk) \neq 2, 3$) they give a formula, in terms of a_6, for four values $y_1, \ldots, y_4$ such that the equation $x^3 + a_6 = y_i^2$ has a solution $x \in \Bbbk$ for some $1 \leq i \leq 4$. This method therefore produces at most 12 points on any given curve.

Skałba [569] gave results for general curves $y^2 = F(x)$ where $F(x) = x^3 + a_4 x + a_6$. This method can give more than a fixed number of points for any given curve. More precisely, Skałba gives explicit rational functions $X_i(t) \in \mathbb{Q}(t)$ for $1 \leq i \leq 3$ such that there is a rational function $U(t) \in \mathbb{Q}(t)$ such that

$$F(X_1(t^2))F(X_2(t^2))F(X_3(t^2)) = U(t)^2.$$

In other words, Skałba gives a rational map from $\mathbb{A}^1$ to the variety $F(x_1)F(x_2)F(x_3) = u^2$. Evaluating at $t \in \mathbb{F}_p$, where $p > 3$ is prime, it follows that at least one of the $F(X_i(t^2))$ is a square in $\mathbb{F}_p^*$. One can therefore find a point on E by taking square roots. Note that efficient algorithms for computing square roots modulo p are randomised in general. Skałba suggests avoiding this problem by assuming that the required quadratic non-residue has been precomputed (as in Exercise 2.9.6).

Shallue and van de Woestijne [545] improve upon Skałba's algorithm in several ways. First, and most significantly, they show that a deterministic sampling algorithm does not require a quadratic non-residue modulo p. They achieve this by cleverly using all three values $F(X_1(t^2)), F(X_2(t^2))$ and $F(X_3(t^2))$. In addition, they give a simpler rational map (see Exercise 11.4.12) from $\mathbb{A}^1$ to the variety $F(x_1)F(x_2)F(x_3) = u^2$, and handle the characteristic 2 and 3 cases.

Exercise 11.4.12. (Shallue and van de Woestijne [545]) Let $F(x) = x^3 + Ax + B$ and $H(u,v) = u^2 + uv + v^2 + A(u+v) + B$. Let $V : F(x_1)F(x_2)F(x_3) = u^2$ and let $S : y^2H(u,v) = -F(u)$. Show that the map $\psi(u,v,y) \to (v, -A-u-v, u+y^2, F(u+y^2)H(u,v)/y)$ is a rational map from S to V. Let $p > 3$ be prime. Fix $u \in \mathbb{F}_p$ such that $F(u) \neq 0$ and $3u^2 + 2Au + 4B - A^2 \neq 0$. Show that the surface S for this fixed value of u is

$$[y(v + u/2 + A/2)]^2 + [3u^2/4 + Au/2 + B - A^2/4]y^2 = -F(u).$$

Hence, show there is a rational map from $\mathbb{A}^1$ to S and hence a rational map from $\mathbb{A}^1$ to V.

It is worth noting that there can be no rational map $\phi : \mathbb{P}^1 \to C$ when C is a curve of genus at least 1. This follows from the Hurwitz genus formula: if the map has degree d then we have $-2 = 2g(\mathbb{P}^1) - 2 = d(2g(C) - 2) + R \geq 0$ where R is a positive integer counting the ramification, which is a contradiction. The above maps do not contradict this fact. They are not rational maps from $\mathbb{P}^1$ (or $\mathbb{A}^1$) to an elliptic curve; there is always one part of the function (such as computing a square-root or cube-root) that is not a rational map.

Icart [303] has given a simpler map for elliptic curves $y^2 = x^3 + Ax + B$ over $\mathbb{F}_q$ when $q \equiv 2 \pmod 3$. Let $u \in \mathbb{F}_q$. Define

$$v = (3A - u^4)/(6u)\ , \quad x = \left(v^2 - B - u^6/27\right)^{1/3} + u^2/3 \quad \text{and} \quad y = ux + v \qquad (11.4)$$

where the cube root is computed by exponentiating to the power $(2q-1)/3 \equiv 3^{-1} \pmod{(q-1)}$.

Exercise 11.4.13. Verify that the point (x,y) of equation (11.4) is a point on $E : y^2 = x^3 + Ax + B$ over $\mathbb{F}_q$. Show that, given a point (x,y) on an elliptic curve E over $\mathbb{F}_q$ as above, one can efficiently compute $u \in \mathbb{F}_q$, if it exists, such that the process of equation (11.4) gives the point (x,y).

For elliptic or hyperelliptic curves of the form $y^2 = F(x)$ where $F(x) = x^n + Ax + B$ or $F(x) = x^n + Ax^2 + B$, Ulas [612] gives a rational map from $\mathbb{A}^1$ to the variety $F(x_1)F(x_2)F(x_3) = u^2$. Hence, it is possible to deterministically find points on hyperelliptic curves of this form.

Exercise 11.4.14. (Ulas) Let $F(x) = x^3 + Ax + B$ and define

$$\begin{array}{rclrcl} X_1(t,u) & = & u, & X_2(t,u) & = & -B/A(t^6F(u)^3 - 1)/(t^6F(u)^3 - t^2F(u)), \\ X_3(t,u) & = & t^2F(u)X_2(t,u), & U(t,u) & = & t^3F(u)^2F(X_2(t,u)). \end{array}$$

Show that $U(t,u)^2 = F(X_1(t,u))F(X_2(t,u))F(X_3(t,u))$.
[Hint: Use a computer algebra package.]

Note that, for curves of genus 2 or more, a related computational problem is to deterministically find rational degree 0 divisor classes. A simple solution is to generate two points $P, Q \in C(\mathbb{F}_q)$ using the above methods and let $D = (P) - (Q)$. More care is needed to ensure that the divisor classes are distributed uniformly. We finish with an exercise that shows that a natural method to generate rational divisor classes is not useful for this application.

Exercise 11.4.15. Let $y^2 = F(x)$ be a hyperelliptic curve of genus $g \geq 2$ over $\mathbb{F}_p$ with $p > 2$ in imaginary model. Denote by P_0 the point at infinity. Let $x \in \mathbb{F}_p$ and suppose one has computed $y = \sqrt{F(x_0)} \in \mathbb{F}_{p^2}$. Let $P = (x,y)$. Show that if $y \notin \mathbb{F}_p$ then $D = (P) + (\sigma(P)) - 2(P_0)$ is principal, where σ is the non-trivial element of $\mathrm{Gal}(\mathbb{F}_{p^2}/\mathbb{F}_p)$.

In most cryptographic applications we are interested in sampling from subgroups of $E(\mathbb{F}_q)$ of prime order r. As mentioned earlier, the simplest way to transform elements sampled randomly in $E(\mathbb{F}_q)$ into random elements of the subgroup is to exponentiate to the power $\#E(\mathbb{F}_q)/r$ (assuming that $r \| \#E(\mathbb{F}_q)$).

11.4.3 Hashing to Algebraic Groups

Recall that sampling from algebraic groups is the task of selecting group elements uniformly at random. On the other hand, a hash function $H : \{0,1\}^l \to G(\mathbb{F}_q)$ is a deterministic algorithm that takes an input $\mathsf{m} \in \{0,1\}^l$ and outputs a group element. It is required that the output distribution of H, corresponding to the uniform distribution of the message space, is close to uniform in the group $G(\mathbb{F}_q)$. The basic idea is to use m as the randomness required by the sampling algorithm.

Recall that a hash function is also usually required to satisfy some security requirements, such as collision-resistance. This is usually achieved by first applying a collision-resistant hash function $H' : \{0,1\}^l \to \{0,1\}^l$ and setting $\mathsf{m}' = H'(\mathsf{m})$. In this section we are only concerned with the problem of using m' as input to a sampling algorithm.

The first case to consider is hashing to $\mathbb{F}_p^*$. If $p > 2^l + 1$ then we are in trouble, since one cannot get uniform coverage of a set of size $p-1$ using fewer than $p-1$ elements. This shows that we always need $l > \log_2(\#G(\mathbb{F}_q))$ (though, in some applications, it might be possible to still have a useful cryptographic system even when the image of the hash function is a subset of the group).

Example 11.4.16. Suppose $2^l > p$ and $\mathsf{m} \in \{0,1\}^l$. The method of Example 11.4.2 gives output close to uniform (at least, if $l - \log_2(p)$ is reasonably large).

Exercise 11.4.17. Let $q = p^n < 2^l$. Give a hash function $H : \{0,1\}^l \to \mathbb{F}_q$.

It is relatively straightforward to turn the algorithms of Example 11.4.8 and Exercise 11.4.9 into hash functions. In the elliptic curve case there is a growing literature on transforming a sampling algorithm into a hash function. We do not give the details.

11.4.4 Hashing from Algebraic Groups

In some applications it is also necessary to have a hash function $H : G(\mathbb{F}_q) \to \{0,1\}^l$ where G is an algebraic group. Motivation for this problem is given in the discussion of key derivation functions in Section 20.2.3. A framework for problems of this type is **randomness extraction**. It is beyond the scope of this book to give a presentation of this topic, but some related results are given in Sections 21.7 and 21.6.

11.5 Determining Group Structure and Computing Generators for Elliptic Curves

Since $\mathbb{F}_q^*$ is cyclic, it follows that all subgroups of finite fields and tori are cyclic. However, elliptic curves and divisor class groups of hyperelliptic curves can be non-cyclic. Determining the group structure and a set of generators for an algebraic group $G(\mathbb{F}_q)$ can be necessary for some applications. It is important to remark that solutions to these problems are not expected to exist if the order $N = \#G(\mathbb{F}_q)$ is not known, or if the factorisation of N is not known.

Let E be an elliptic curve over $\mathbb{F}_q$ and let $N = \#E(\mathbb{F}_q)$. If N has no square factors then $E(\mathbb{F}_q)$ is isomorphic as a group to $\mathbb{Z}/N\mathbb{Z}$. If $r^2 \| N$ then there could be a point of

order r^2 or two "independent" points of order r (i.e., $E(\mathbb{F}_q)$ has a non-cyclic subgroup of order r^2 but exponent r).

The Weil pairing (see Section 26.2) can be used to determine the group structure of an elliptic curve. Let r be a prime and $P, Q \in E(\mathbb{F}_q)$ of order r. The key fact is that the Weil pairing is alternating and so $e_r(P, P) = 1$. It follows from the non-degeneracy of the pairing that $e_r(P, Q) = 1$ if and only if $Q \in \langle P \rangle$. The Weil pairing also shows that one can only have two independent points when r divides $(q - 1)$.

Given the factorisation of $\gcd(q - 1, \#E(\mathbb{F}_q))$, the group structure can be determined using a randomised algorithm due to Miller [427, 429]. We present this algorithm in Figure 10. Note that the algorithm of Theorem 2.15.10 is used in lines 7 and 10. The expected running time is polynomial, but we refer to Miller [429] for the details.

Algorithm 10 Miller's algorithm for group structure

INPUT: $E/\mathbb{F}_q$, $N_0, N_1 \in \mathbb{N}$ and the factorisation of N_0, where $\#E(\mathbb{F}_q) = N_0 N_1$, $\gcd(N_1, q - 1) = 1$ and all primes dividing N_0 divide $q - 1$

OUTPUT: Integers m and n such that $E(\mathbb{F}_q) \cong (\mathbb{Z}/m\mathbb{Z}) \times (\mathbb{Z}/n\mathbb{Z})$ as a group

1: Write $N_0 = \prod_{i=1}^{k} l_i^{e_i}$ where $l_1, \ldots, l_k$ are distinct primes
2: For all $1 \leq i \leq k$ such that $e_i = 1$ set $N_0 = N_0/l_i, N_1 = N_1 l_i$
3: $m = 1$, $n = 1$
4: **while** $mn \neq N_0$ **do**
5: Choose random points $P', Q' \in E(\mathbb{F}_q)$
6: $P = [N_1]P'$, $Q = [N_1]Q'$
7: Find the exact orders m' and n' of P and Q
8: $n = \mathrm{lcm}(m', n')$
9: $\alpha = e_n(P, Q)$
10: Let m be the exact order of α in $\mu_n = \{z \in \mathbb{F}_q^* : z^n = 1\}$
11: **end while**
12: **return** m and nN_1

Exercise 11.5.1. Show that Algorithm 10 is correct.

Exercise 11.5.2. Modify Algorithm 10 so that it outputs generators for $E(\mathbb{F}_q)$.

Exercise 11.5.3. This exercise will determine the expected number of iterations of Algorithm 10. Let $N_0 = \prod_{i=1}^{k} l_i^{e_i}$ with $e_i > 1$ for all $1 \leq i \leq k$ where $l_1, \ldots, l_k$ are distinct primes.

Let $(l, e) = (l_i, e_i)$ for some $1 \leq i \leq k$. Write $E(\mathbb{F}_q)[l^e]$ for the subgroup of $E(\mathbb{F}_q)$ consisting of elements of order dividing l^e. This group may or may not be cyclic. Show that the probability that a pair of randomly chosen group elements generate $E(\mathbb{F}_q)[l^e]$ is at least

$$\left(1 - \tfrac{1}{l}\right)\left(1 - \tfrac{1}{l^2}\right).$$

Now, show that the probability of success overall in one iteration is at least

$$\frac{\varphi(N_0)}{N_0} \prod_{l | N_0} \left(1 - \tfrac{1}{l^2}\right)$$

where $\varphi(n)$ is the Euler phi function. Finally, apply Theorem A.3.1 and the fact that $\zeta(2) = \pi^2/6$ (this is the Riemann zeta function) to show that the algorithm requires $O(\log(\log(q)))$ iterations.

Kohel and Shparlinski [352] give a deterministic algorithm to compute the group structure and to find generators for $E(\mathbb{F}_q)$. Their algorithm requires $O(q^{1/2+\epsilon})$ bit operations.

11.6 Testing Subgroup Membership

In many cryptographic protocols it is necessary to verify that the elements received really do correspond to group elements with the right properties. There are a variety of attacks that can be performed otherwise, some of which are briefly mentioned in Section 20.4.2.

The first issue is whether a binary string corresponds to an element of the "parent group" $G(\mathbb{F}_q)$. This is usually easy to check when $G(\mathbb{F}_q) = \mathbb{F}_q^*$. In the case of elliptic curves one must parse the bitstring as a point (x, y) and determine that (x, y) does satisfy the curve equation.

The more difficult problem is testing whether a group element g lies in the desired subgroup. For example, if $r \| \#G(\mathbb{F}_q)$ and we are given a group element g, to ensure that g lies in the unique subgroup of order r one can compute g^r and check if this is the identity. Efficient exponentiation algorithms can be used, but the computational cost is still significant. In some situations one can more efficiently test subgroup membership. One notable case is when $\#G(\mathbb{F}_q)$ is prime, this is one reason why elliptic curves of prime order are so convenient for cryptography.

Example 11.6.1. Let $p = 2r + 1$ be a **safe prime** or **Sophie-Germain prime** (i.e., r and p are primes). Then an element $g \in \mathbb{F}_p^*$ lies in the subgroup of order r if and only if $(\frac{g}{p}) = 1$. Note that one can compute $(\frac{g}{p}) = 1$ in $O(\log(p)^2)$ bit operations, whereas computing g^r in this case requires $O(\log(p)M(\log(p)))$ bit operations. However, in practice computing the Legendre symbol may not be significantly faster than computing g^r. Also, there are other performance problems from using very large subgroups of F_p^* (for example, signature size).

Exercise 11.6.2. (King [339]) Let E be an elliptic curve over $\mathbb{F}_q$ such that $\#E(\mathbb{F}_q) = 2^m r$ where m is small and r is prime. Show how to use point halving (see Exercise 9.1.4) to efficiently determine whether a point $P \in E(\mathbb{F}_q)$ has order dividing r.

An alternative way to prevent attacks due to elements of incorrect group order is to "force" all group elements to lie in the required subgroup by exponentiating to a **cofactor** (such as $\#G(\mathbb{F}_q)/r$). When the cofactor is small this can be a more efficient way to deal with the problem than testing subgroup membership, though one must ensure the cryptographic system can function correctly in this setting.

With algebraic group quotients represented using traces (i.e., LUC and XTR) one represents a finite field element using a trace. This value corresponds to a valid element of the extension field only if certain conditions hold. In the case of LUC we represent $g \in G_{2,p}$, where p is prime, by the trace $V = \mathrm{Tr}(g)$. A value V corresponds to an element of $G_{2,q}$ if and only if the quadratic polynomial $(x-g)(x-g^p) = x^2 - Vx + 1$ is irreducible (in other words, if $(\frac{V^2-4}{p}) = -1$). Similarly, in XTR one needs to check whether the polynomial $x^3 - tx^2 + t^p x - 1$ is irreducible; Lenstra and Verheul [376] have given efficient algorithms to do this. Section 4 of [376] also discusses subgroup attacks in the context of XTR and countermeasures in this context.

11.7 Elliptic Curve Point Compression

When elements of an algebraic group are transmitted one often wants to minimise the number of bits sent. Indeed, one of the main motivations for torus/trace cryptography is compression of elements. With elliptic curves it is obvious that, given a point (x, y), one only needs to send x together with a single bit to specify the choice of y. To obtain the point the receiver then has to solve a quadratic equation over the finite field. An

alternative, suitable only for some applications, is to work in the algebraic group quotient corresponding to elliptic curve arithmetic using x-coordinates only. We briefly mention further tricks in the elliptic curve setting.

Example 11.7.1. (Seroussi [540]) Let $E : y^2+xy = x^3+a_2x^2+a_6$ be an ordinary elliptic curve over $\mathbb{F}_{2^n}$, so that $\#E(\mathbb{F}_{2^n})$ is even. Let $P \in E(\mathbb{F}_{2^n})$ be a point of odd order, so that $P = [2]Q$ for some point $Q \in E(\mathbb{F}_{2^n})$.

Recall from Exercise 9.1.4 that $P = (x_P, y_P) \in E(\mathbb{F}_{2^n})$ is of the form $P = [2]Q$ for some $Q \in E(\mathbb{F}_{2^n})$ if and only if $\mathrm{Tr}_{\mathbb{F}_{2^n}/\mathbb{F}_2}(x_P) = \mathrm{Tr}_{\mathbb{F}_{2^n}/\mathbb{F}_2}(a_2)$. Hence, the trace of x_P is public knowledge.

Suppose $\mathbb{F}_{2^n}$ is represented using a normal basis, so that $\mathrm{Tr}_{\mathbb{F}_{2^n}/\mathbb{F}_2}(x_P)$ is congruent modulo 2 to the Hamming weight of the representation of x_P. Then one can compress (x_P, y_P) by discarding y_P, removing the least significant bit of the representation of x_P and adding a bit to determine the choice of y_P. Hence, one needs n bits to send (x_P, y_P).

Exercise 11.7.2.★ (King [339]) Let the notation be as in Example 11.7.1 and suppose further that $\mathrm{Tr}(a_2) = 0$, where Tr denotes $\mathrm{Tr}_{\mathbb{F}_{2^n}/\mathbb{F}_2}$. Let $(x_P, y_P) \in E(\mathbb{F}_{2^n})$ be a point of odd order. Show that $\mathrm{Tr}(a_6/x_P^2) = 0$ and so $\mathrm{Tr}(\sqrt{a_6}/x_P) = 0$. Show that $(0, \sqrt{a_6})$ has order 2 and that this point can be halved (in other words, there is a point $R \in E(\mathbb{F}_{2^n})$ such that $(0, \sqrt{a_6}) = [2]R$). Show that $(x_P, y_P)+(0, \sqrt{a_6}) = (\sqrt{a_6}/x_P, (y_P\sqrt{a_6}+a_6)/x_P^2+\sqrt{a_6}(1+1/x_P))$ and hence deduce that this point can also be halved.

Hence, show that one can send $(x_P, y_P) \in E(\mathbb{F}_{2^n})$ using only $n-1$ bits by sending x_P with one bit omitted when $\mathrm{Tr}(y_P/x_P) = 0$ or $\sqrt{a_6}/x_P$ with one bit omitted when $\mathrm{Tr}(y_P/x_P) = 1$.

Exercise 11.7.3.★ (Galbraith and Eagle) Let E be an elliptic curve over $\mathbb{F}_2$ and let $P = (x_P, y_P) \in E(\mathbb{F}_{2^n})$. Explain how one might compress P to roughly $n - \log_2(n)$ bits on average.

We refer to Section 14.2 of [16] for the details of divisor compression for hyperelliptic curves.