# Chapter 13

# Basic Discrete Logarithm Algorithms



This chapter is about algorithms to solve the discrete logarithm problem (DLP) and some variants of it. We focus mainly on deterministic methods that work in any group; later chapters will present the Pollard rho and kangaroo methods, and index calculus algorithms. In this chapter we also present the concept of generic algorithms and prove lower bounds on the running time of a generic algorithm for the DLP. The starting point is the following definition (already given as Definition 2.1.1).

**Definition 13.0.1.** Let $G$ be a group written in multiplicative notation. The **discrete logarithm problem** (**DLP**) is: Given $g, h \in G$ to find $a$, if it exists, such that $h = g^a$. We sometimes denote $a$ by $\log_g(h)$.

As discussed after Definition 2.1.1, we intentionally do not specify a distribution on $g$ or $h$ or $a$ above, although it is common to assume that $g$ is sampled uniformly at random in $G$ and $a$ is sampled uniformly from $\{1, \ldots, \#G\}$.

Typically $G$ will be an algebraic group over a finite field $\mathbb{F}_q$ and the order of $g$ will be known. If one is considering cryptography in an algebraic group quotient then we assume that the DLP has been lifted to the covering group $G$. A solution to the DLP exists if and only if $h \in \langle g \rangle$ (i.e., $h$ lies in the subgroup generated by $g$). We have discussed methods to test this in Section 11.6.

**Exercise 13.0.2.** Consider the discrete logarithm problem in the group of integers modulo $p$ under **addition**. Show that the discrete logarithm problem in this case can be solved in polynomial-time.

Exercise 13.0.2 shows there are groups for which the DLP is easy. The focus in this book is on algebraic groups for which the DLP seems to be hard.

**Exercise 13.0.3.** Let $N$ be composite. Define the discrete logarithm problem DLP-MOD-N in the multiplicative group of integers modulo $N$. Show that FACTOR $\leq_R$ DLP-MOD-N.

Exercise 13.0.3 gives some evidence that cryptosystems based on the DLP should be at least as secure as cryptosystems based on factoring.

## 13.1 Exhaustive Search

The simplest algorithm for the DLP is to sequentially compute $g^a$ for $0 \leq a < r$ and test equality of each value with $h$. This requires at most $r-2$ group operations and $r$ comparisons.

**Exercise 13.1.1.** Write pseudocode for the exhaustive search algorithm for the DLP and verify the claims about the worst-case number of group operations and comparisons.

If the cost of testing equality of group elements is $O(1)$ group operations then the worst-case running time of the algorithm is $O(r)$ group operations. It is natural to assume that testing equality is always $O(1)$ group operations, and this will always be true for the algebraic groups considered in this book. However, as Exercise 13.1.2 shows, such an assumption is not entirely trivial.

**Exercise 13.1.2.** Suppose projective coordinates are used for elliptic curves $E(\mathbb{F}_q)$ to speed up the group operations in the exhaustive search algorithm. Show that testing equality between a point in projective coordinates and a point in affine or projective coordinates requires at least one multiplication in $\mathbb{F}_q$ (and so this cost is not linear). Show that, nevertheless, the cost of testing equality is less than the cost of a group operation.

For the rest of this chapter we assume that groups are represented in a compact way and that operations involving the representation of the group (e.g., testing equality) all cost less than the cost of one group operation. This assumption is satisfied for all the algebraic groups studied in this book.

## 13.2 The Pohlig-Hellman Method

Let $g$ have order $N$ and let $h = g^a$, so that $h$ lies in the cyclic group generated by $g$. Suppose $N = \prod_{i=1}^{n} l_i^{e_i}$. The idea of the Pohlig-Hellman[1] method [482] is to compute $a$ modulo the prime powers $l_i^{e_i}$ and then recover the solution using the Chinese remainder theorem. The main ingredient is the following group homomorphism, which reduces the discrete logarithm problem to subgroups of prime power order.

**Lemma 13.2.1.** *Suppose $g$ has order $N$ and $l^e \mid N$. The function*

$$\Phi_{l^e}(g) = g^{N/l^e}$$

*is a group homomorphism from $\langle g \rangle$ to the unique cyclic subgroup of $\langle g \rangle$ of order $l^e$. Hence, if $h = g^a$ then*

$$\Phi_{l^e}(h) = \Phi_{l^e}(g)^{a \pmod{l^e}}.$$

[1] The paper [482] is authored by Pohlig and Hellman and so the method is usually referred to by this name, although R. Silver, R. Schroeppel, H. Block, and V. Nechaev also discovered it.

**Exercise 13.2.2.** Prove Lemma 13.2.1.

Using $\Phi_{l^e}$ one can reduce the DLP to subgroups of prime power order. To reduce the problem to subgroups of prime order we do the following: Suppose $g_0$ has order $l^e$ and $h_0 = g_0^a$ then we can write $a = a_0 + a_1 l + \cdots a_{e-1} l^{e-1}$ where $0 \leq a_i < l$. Let $g_1 = g_0^{l^{e-1}}$. Raising to the power $l^{e-1}$ gives

$$h_0^{l^{e-1}} = g_1^{a_0}$$

from which one can find $a_0$ by trying all possibilities (or using baby-step-giant-step or other methods).

To compute $a_1$ we define $h_1 = h_0 g_0^{-a_0}$ so that

$$h_1 = g_0^{a_1 l + a_2 l^2 + \cdots a_{e-1} l^{e-1}}.$$

Then $a_1$ is obtained by solving

$$h_1^{l^{e-2}} = g_1^{a_1}$$

To obtain the next value we set $h_2 = h_1 g_0^{-l a_1}$ and repeat. Continuing gives the full solution modulo $l^e$. Once $a$ is known modulo $l_i^{e_i}$ for all $l_i^{e_i} \| N$ one computes $a$ using the Chinese remainder theorem. The full algorithm (in a slightly more efficient variant) is given in Algorithm 13.

**Algorithm 13** Pohlig-Hellman algorithm

INPUT: $g, h = g^a$, $\{(l_i, e_i) : 1 \leq i \leq n\}$ such that order of $g$ is $N = \prod_{i=1}^n l_i^{e_i}$
OUTPUT: $a$

1: Compute $\{g^{N/l_i^{f_i}}, h^{N/l_i^{f_i}} : 1 \leq i \leq n, 1 \leq f_i \leq e_i\}$
2: **for** $i = 1$ to $n$ **do**
3: $a_i = 0$
4: **for** $j = 1$ to $e_i$ **do** ▷ Reducing DLP of order $l_i^{e_i}$ to cyclic groups
5: Let $g_0 = g^{N/l_i^j}$ and $h_0 = h^{N/l_i^j}$ ▷ These were already computed in line 1
6: Compute $u = g_0^{-a_i}$ and $h_0 = h_0 u$
7: **if** $h_0 \neq 1$ **then**
8: Let $g_0 = g^{N/l_i}$, $b = 1$, $T = g_0$ ▷ Already computed in line 1
9: **while** $h_0 \neq T$ **do** ▷ Exhaustive search
10: $b = b + 1$, $T = T g_0$
11: **end while**
12: $a_i = a_i + b l_i^{j-1}$
13: **end if**
14: **end for**
15: **end for**
16: Use Chinese remainder theorem to compute $a \equiv a_i \pmod{l_i^{e_i}}$ for $1 \leq i \leq n$
17: **return** $a$

**Example 13.2.3.** Let $p = 19, g = 2$ and $h = 5$. The aim is to find an integer $a$ such that $h \equiv g^a \pmod{p}$. Note that $p - 1 = 2 \cdot 3^2$. We first find $a$ modulo 2. We have $(p-1)/2 = 9$ so define $g_0 = g^9 \equiv -1 \pmod{19}$ and $h_0 = h^9 \equiv 1 \pmod{19}$. It follows that $a \equiv 0 \pmod{2}$.

Now we find $a$ modulo 9. Since $(p-1)/9 = 2$ we first compute $g_0 = g^2 \equiv 4 \pmod{19}$ and $h_0 \equiv h^2 \equiv 6 \pmod{19}$. To get information modulo 3 we compute (this is a slight change of notation from Algorithm 13)

$$g_1 = g_0^3 \equiv 7 \pmod{19} \text{ and } h_0^3 \equiv 7 \pmod{19}.$$

It follows that $a \equiv 1 \pmod 3$. To get information modulo 9 we remove the modulo 3 part by setting $h_1 = h_0/g_0 = 6/4 \equiv 11 \pmod{19}$. We now solve $h_1 \equiv g_1^{a_1} \pmod{19}$, which has the solution $a_1 \equiv 2 \pmod 3$. It follows that $a \equiv 1 + 3 \cdot 2 \equiv 7 \pmod 9$.

Finally, by the Chinese remainder theorem we obtain $a \equiv 16 \pmod{18}$.

**Exercise 13.2.4.** Let $p = 31$, $g = 3$ and $h = 22$. Solve the discrete logarithm problem of $h$ to the base $g$ using the Pohlig-Hellman method.

We recall that an integer is $B$-smooth if all its prime factors are at most $B$.

**Theorem 13.2.5.** *Let $g \in G$ have order $N$. Let $B \in \mathbb{N}$ be such that $N$ is $B$-smooth Then Algorithm 13 solves the DLP in $G$ using $O(\log(N)^2 + B\log(N))$ group operations.*[2]

**Proof:** One can factor $N$ using trial division in $O(BM(\log(N)))$ bit operations, where $M(n)$ is the cost of multiplying $n$-bit integers. We assume that $M(\log(N))$ is $O(1)$ group operations (this is true for all the algebraic groups of interest in this book). Hence, we may assume that the factorisation of $N$ is known.

Computing all $\Phi_{l_i^{e_i}}(g)$ and $\Phi_{l_i^{e_i}}(h)$ can be done naively in $O(\log(N)^2)$ group operations, but we prefer to do it in $O(\log(N)\log\log(N))$ group operations using the method of Section 2.15.1.

Lines 5 to 13 run $\sum_{i=1}^n e_i = O(\log(N))$ times and, since each $l_i \geq 2$, we have $\sum_{i=1}^n e_i \leq \log_2(N)$. The computation of $u$ in line 6 requires $O(e_i \log(l_i))$ group operations. Together this gives a bound of $O(\log(N)^2)$ group operations to the running time. (Note that when $N = 2^e$ then the cost of these lines is $e^2\log(2) = O(\log(N)^2)$ group operations.)

Solving each DLP in a cyclic group of order $l_i$ using naive methods requires $O(l_i)$ group operations (this can be improved using the baby-step-giant-step method). There are $\leq \log_2(N)$ such computations to perform, giving $O(\log(N)B)$ group operations.

The final step is to use the Chinese remainder theorem to compute $a$, requiring $O(\log(N)M(\log(N)))$ bit operations, which is again assumed to cost at most $O(\log(N))$ group operations. □

Due to this method, small primes give no added security in discrete logarithm systems. Hence one generally uses elements of prime order $r$ for cryptography.

**Exercise 13.2.6.** Recall the Tonelli-Shanks algorithm for computing square roots modulo $p$ from Section 2.9. A key step of the algorithm is to find a solution $j$ to the equation $b = y^{2j} \pmod p$ where $y$ has order $2^e$. Write down the Pohlig-Hellman method to solve this problem. Show that the complexity is $O(\log(p)^2 M(\log(p)))$ bit operations.

**Exercise 13.2.7.** Let $B \in \mathbb{N}_{>3}$. Let $N = \prod_{i=1}^n x_i$ where $2 \leq x_i \leq B$. Prove that $\sum_{i=1}^n x_i \leq B\log(N)/\log(B)$.

Hence, show that the Pohlig-Hellman method performs $O(\log(N)^2 + B\log(N)/\log(B))$ group operations.

**Remark 13.2.8.** As we will see, replacing exhaustive search by the baby-step-giant-step algorithm improves the complexity to $O(\log(N)^2 + \sqrt{B}\log(N)/\log(B))$ group operations (at the cost of more storage).

Algorithm 13 can be improved, when there is a prime power $l^e$ dividing $N$ with $e$ large, by structuring it differently. Section 11.2.3 of Shoup [556] gives a method to compute the DLP in a group of order $l^e$ in $O(e\sqrt{l} + e\log(e)\log(l))$ group operations (this is using baby-step-giant-step rather than exhaustive search). Algorithm 1 and Corollary 1 of Sutherland [598] give an algorithm that requires

$$O(e\sqrt{l} + e\log(l)\log(e)/\log(\log(e))) \tag{13.1}$$

[2] By this we mean that the constant implicit in the $O(\cdot)$ is independent of $B$ and $N$.

group operations. Sutherland also considers non-cyclic groups.

If $N$ is $B$-smooth then summing the improved complexity statements over the prime powers dividing $N$ gives

$$O(\log(N)\sqrt{B}/\log(B) + \log(N)\log(\log(N))) \tag{13.2}$$

group operations for the DLP (it is not possible to have a denominator of $\log(\log(\log(N)))$ since not all the primes dividing $N$ necessarily appear with high multiplicity).

## 13.3 Baby-Step-Giant-Step (BSGS) Method

This algorithm, usually credited to Shanks[3], exploits an idea called the time/memory tradeoff. Suppose $g$ has prime order $r$ and that $h = g^a$ for some $0 \leq a < r$. Let $m = \lceil \sqrt{r} \rceil$. Then there are integers $a_0, a_1$ such that $a = a_0 + ma_1$ and $0 \leq a_0, a_1 < m$. It follows that

$$g^{a_0} = h(g^{-m})^{a_1}$$

and this observation leads to Algorithm 14. The algorithm requires storing a large list of values and it is important, in the second stage of the algorithm, to be able to efficiently determine whether or not an element lies in the list. There are a number of standard solutions to this problem including using binary trees, hash tables, or sorting the list after line 7 of the algorithm (see, for example, parts II and III of [146] or Section 6.3 of [317]).

**Algorithm 14** Baby-step-giant-step (BSGS) algorithm

INPUT: $g, h \in G$ of order $r$
OUTPUT: $a$ such that $h = g^a$, or $\perp$

1: $m = \lceil \sqrt{r} \rceil$
2: Initialise an easily searched structure (such as a binary tree or a hash table) $L$
3: $x = 1$
4: **for** $i = 0$ to $m$ **do** ▷ Compute baby steps
5: store $(x, i)$ in $L$, easily searchable on the first coordinate
6: $x = xg$
7: **end for**
8: $u = g^{-m}$
9: $y = h$, $j = 0$
10: **while** $(y, \star) \notin L$ **do** ▷ Compute giant steps
11: $y = yu$, $j = j + 1$
12: **end while**
13: **if** $\exists (x, i) \in L$ such that $x = y$ **then**
14: **return** $i + mj$
15: **else**
16: **return** $\perp$
17: **end if**

Note that the BSGS algorithm is deterministic. The algorithm also solves the decision problem (is $h \in \langle g \rangle$?) though, as discussed in Section 11.6, there are usually faster solutions to the decision problem.

**Theorem 13.3.1.** *Let $G$ be a group of order $r$. Suppose that elements of $G$ are represented using $O(\log(r))$ bits and that the group operations can be performed in $O(\log(r)^2)$ bit*

[3] Nechaev [452] states it was known to Gel'fond in 1962.

*operations. The BSGS algorithm for the DLP in $G$ has running time $O(\sqrt{r}\log(r)^2)$ bit operations. The algorithm requires $O(\sqrt{r}\log(r))$ bits of storage.*

**Proof:** The algorithm computes $\sqrt{r}$ group operations for the baby steps. The cost of inserting each group element into the easily searched structure is $O(\log(r)^2)$ bit operations, since comparisons require $O(\log(r))$ bit operations (this is where the assumption on the size of element representations appears). The structure requires $O(\sqrt{r}\log(r))$ bits of storage.

The computation of $u = g^{-m}$ in line 8 requires $O(\log(r))$ group operations.

The algorithm needs one group operation to compute each giant step. Searching the structure takes $O(\log(r)^2)$ bit operations. In the worst case one has to compute $m$ giant steps. The total running time is therefore $O(\sqrt{r}\log(r)^2)$ bit operations. □

The storage requirement of the BSGS algorithm quickly becomes prohibitive. For example, one can work with primes $r$ such that $\sqrt{r}$ is more than the number of fundamental particles in the universe!

**Remark 13.3.2.** When solving the DLP it is natural to implement the group operations as efficiently as possible. For example, when using elliptic curves it would be tempting to use a projective representation for group elements (see Exercise 13.1.2). However this is not suitable for the BSGS method (or the rho and kangaroo methods) as one cannot efficiently detect a match $y \in L$ when there is a non-unique representation for the group element $y$.

**Exercise 13.3.3.** On average, the baby-step-giant-step algorithm finds a match after half the giant steps have been performed. The average-case running time of the algorithm as presented is therefore approximately $1.5\sqrt{r}$ group operations. Show how to obtain an algorithm that requires, in the average case, approximately $\sqrt{2r}$ group operations and $\sqrt{r/2}$ group elements of storage.

**Exercise 13.3.4.** (Pollard [488]) A variant of the baby-step-giant-step algorithm is to compute the baby steps and giant steps in parallel, storing the points together in a single structure. Show that if $x$ and $y$ are chosen uniformly in the interval $[0, r] \cap \mathbb{Z}$ then the expected value of $\max\{x, y\}$ is approximately $\frac{2}{3}r$. Hence, show that the average-case running time of this variant of the baby-step-giant-step algorithm is $\frac{4}{3}\sqrt{r}$ group operations.

Chateauneuf, Ling and Stinson [129] have studied a combinatorial abstraction that would lead to an optimal baby-step-giant-step algorithm. However their model minimises the total number of *exponentiations* in the group, rather than the total number of group operations, and so is not faster in practice than the methods in this section.

**Exercise 13.3.5.** Design a variant of the BSGS method that requires $O(r/M)$ group operations if the available storage is only for $M < \sqrt{r}$ group elements.

**Exercise 13.3.6.** (DLP in an interval) Suppose one is given $g$ of order $r$ in a group $G$ and integers $0 \leq b, w < r$. The DLP in an interval of length $w$ is: Given $h \in \langle g \rangle$ such that $h = g^a$ for some $b \leq a < b + w$, to find $a$. Give a baby-step-giant-step algorithm to find $a$ in average-case $\sqrt{2w}$ group operations and $\sqrt{w/2}$ group elements of storage.

**Exercise 13.3.7.** Suppose one considers the DLP in a group $G$ where computing the inverse $g^{-1}$ is much faster than multiplication in the group. Show how to solve the DLP in an interval of length $w$ using a baby-step-giant-step algorithm in approximately $\sqrt{w}$ group operations in the average case.

**Exercise 13.3.8.** Suppose one is given $g, h \in G$ and $w, b, m \in \mathbb{N}$ such that $h = g^a$ for some integer $a$ satisfying $0 \leq a < w$ and $a \equiv b \pmod{m}$. Show how to reduce this problem to the problem of solving a DLP in an interval of length $\lceil w/m \rceil$.

**Exercise 13.3.9.** Let $g \in G$ have order $N = mr$ where $r$ is prime and $m$ is $\log(N)$-smooth. Suppose $h = g^x$ and $w$ are given such that $0 \leq x < w$. Show how one can compute $x$ by combining the Pohlig-Hellman method and the BSGS algorithm in $O(\log(N)^2 + \sqrt{w/m})$ group operations.

**Exercise 13.3.10.** Suppose one is given $g, h \in G$ and $b_1, b_2, w \in \mathbb{Z}$ ($w > 0$) such that $b_1 + w < b_2$ and $h = g^a$ for some integer $a$ satisfying either $b_1 \leq a < b_1 + w$ or $b_2 \leq a < b_2 + w$. Give an efficient BSGS algorithm for this problem.

**Exercise 13.3.11.** Let $g \in G$ where the order of $g$ and $G$ are not known. Suppose one is given integers $b, w$ such that the order of $g$ lies in the interval $[b, b + w)$. Explain how to use the BSGS method to compute the order of $g$.

**Exercise 13.3.12.★** Suppose one is given an element $g$ of order $r$ and $h_1, \ldots, h_n \in \langle g \rangle$. Show that one can solve the DLP of all $n$ elements $h_i$ to the base $g$ in approximately $2\sqrt{nr}$ group operations (optimised for the worst case) or approximately $\sqrt{2nr}$ (optimised for the average case).

**Exercise 13.3.13.★** Suppose one is given $g \in G$ of order $r$, an integer $w$, and an instance generator for the discrete logarithm problem that outputs $h = g^a \in G$ such that $0 \leq a < w$ according to some known distribution on $\{0, 1, \ldots, w-1\}$. Assume that the distribution is symmetric with mean value $w/2$. Determine the optimal baby-step-giant-step algorithm to solve such a problem.

**Exercise 13.3.14.★** Suppose one is given $g, h \in G$ and $n \in \mathbb{N}$ such that $h = g^a$ where $a$ has a representation as a non-adjacent form NAF (see Section 11.1.1) of length $n < \log_2(r)$. Give an efficient BSGS algorithm to find $a$. What is the running time?

## 13.4 Lower Bound on Complexity of Generic Algorithms for the DLP

This section presents a lower bound for the complexity of the discrete logarithm problem in groups of prime order for algorithms that do not exploit the representation of the group; such algorithms are called generic algorithms. The main challenge is to formally model such algorithms. Babai and Szemerédi [19] defined a black box group to be a group with elements represented (not necessarily uniquely) as binary strings and where multiplication, inversion and testing whether an element is the identity are all performed using oracles. Nechaev [452] used a different model (for which equality testing does not require an oracle query) and obtained $\Omega(\sqrt{r})$ time and space complexity.

Nechaev's paper concerns deterministic algorithms, and so his result does not cover the Pollard algorithms. Shoup [553] gave yet another model for generic algorithms (his model allows randomised algorithms) and proved $\Omega(\sqrt{r})$ time complexity for the DLP and some related problems. This lower bound is often called the **birthday bound** on the DLP.

Shoup's formulation has proven to be very popular with other authors and so we present it in detail. We also describe the model of generic algorithms by Maurer [404]. Further results in this area, and extensions of the generic algorithm model (such as working with groups of composite order, working with groups endowed with pairings, providing

access to decision oracles etc), have been given by Maurer and Wolf [407], Maurer [404], Boneh and Boyen [76, 77], Boyen [95], Rupp, Leander, Bangerter, Dent and Sadeghi [507].

### 13.4.1 Shoup's Model for Generic Algorithms

Fix a constant $t \in \mathbb{R}_{>0}$. When $G$ is the group of points on an elliptic curve of prime order (and log means $\log_2$ as usual) one can take $t = 2$.

**Definition 13.4.1.** An **encoding** of a group $G$ of order $r$ is an injective function $\sigma : G \rightarrow \{0,1\}^{\lceil t \log(r) \rceil}$.

A **generic algorithm** for a computational problem in a group $G$ of order $r$ is a probabilistic algorithm that takes as input $r$ and $(\sigma(g_1), \ldots, \sigma(g_k))$ such that $g_1, \ldots, g_k \in G$ and returns a sequence $(a_1, \ldots, a_l, \sigma(h_1), \ldots, \sigma(h_m))$ for some $a_1, \ldots, a_l \in \mathbb{Z}/r\mathbb{Z}$ and $h_1, \ldots, h_m \in G$ (depending on the computational problem in question). The generic algorithm is given access to a perfect oracle $O$ such that $O(\sigma(g_1), \sigma(g_2))$ returns $\sigma(g_1 g_2^{-1})$.

Note that one can obtain the encoding $\sigma(1)$ of the identity element by $O(\sigma(g_1), \sigma(g_1))$. One can then compute the encoding of $g^{-1}$ from the encoding of $g$ as $O(\sigma(1), \sigma(g))$. Defining $O'(\sigma(g_1), \sigma(g_2)) = O(\sigma(g_1), O(\sigma(1), \sigma(g_2)))$ gives an oracle for multiplication in $G$.

**Example 13.4.2.** A generic algorithm for the DLP in $\langle g \rangle$ where $g$ has order $r$ takes input $(r, \sigma(g), \sigma(h))$ and outputs $a$ such that $h = g^a$. A generic algorithm for CDH (see Definition 20.2.1) takes input $(\sigma(g), \sigma(g^a), \sigma(g^b))$ and outputs $\sigma(g^{ab})$.

In Definition 13.4.1 we insisted that a generic algorithm take as input the order of the group, but this is not essential. Indeed, it is necessary to relax this condition if one wants to consider generic algorithms for, say, $(\mathbb{Z}/N\mathbb{Z})^*$ when $N$ is an integer of unknown factorisation. To do this one considers an encoding function to $\{0,1\}^l$ and it follows that the order $r$ of the group is at most $2^l$. If the order is not given then one can consider a generic algorithm whose goal is to compute the order of a group. Theorem 2.3 and Corollary 2.4 of Sutherland [596] prove an $\Omega(r^{1/3})$ lower bound on the complexity of a generic algorithm to compute the order $r$ of a group, given a bound $M$ such that $\sqrt{M} < r < M$.

### 13.4.2 Maurer's Model for Generic Algorithms

Maurer's formulation of generic algorithms [404] does not use any external representation of group elements (in particular, there are no randomly chosen encodings). Maurer considers a black box containing registers, specified by indices $i \in \mathbb{N}$, that store group elements. The model considers a set of operations and a set of relations. An oracle query $O(op, i_1, \ldots, i_{t+1})$ causes register $i_{t+1}$ to be assigned the value of the $t$-ary operation $op$ on the values in registers $i_1, \ldots, i_t$. Similarly, an oracle query $O(R, i_1, \ldots, i_t)$ returns the value of the $t$-ary relation $R$ on the values in registers $i_1, \ldots, i_t$.

A **generic algorithm** in Maurer's model is an algorithm that takes as input the order of the group (as with Shoup's model, the order of the group can be omitted), makes oracle queries, and outputs the value of some function of the registers (for example, the value of one of the registers; Maurer calls such an algorithm an "extraction algorithm").

**Example 13.4.3.** To define a generic algorithm for the DLP in Maurer's model one imagines a black box that contains in the first register the value 1 (corresponding to $g$) and in the second register the value $a$ (corresponding to $h = g^a$). Note that the black box contains is viewed as containing the additive group $\mathbb{Z}/r\mathbb{Z}$. The algorithm has access

to an oracle $O(+,i,j,k)$ that assigns register $k$ the sum of the elements in registers $i$ and $j$, an oracle $O(-,i,j)$ that assigns register $j$ the inverse of the element in register $i$, and an oracle $O(=,i,j)$ that returns 'true' if and only if registers $i$ and $j$ contain the same group element. The goal of the generic algorithm for the DLP is to output the value of the second register.

To implement the baby-step-giant-step algorithm or Pollard rho algorithm in Maurer's model it is necessary to allow a further oracle that computes a well-ordering relation on the group elements.

We remark that the Shoup and Maurer models have been used to prove the security of cryptographic protocols against adversaries that behave like generic algorithms. Jager and Schwenk [308] have shown that both models are equivalent for this purpose.

### 13.4.3 The Lower Bound

We present the main result of this section using Shoup's model. A similar result can be obtained using Maurer's model (except that it is necessary to either ignore the cost of equality queries or else allow a total order relation on the registers).

We start with a result attributed by Shoup to Schwarz. In this section we only use the result when $k = 1$, but the more general case is used later in the book.

**Lemma 13.4.4.** *Let $F(x_1,\ldots,x_k) \in \mathbb{F}_r[x_1,\ldots,x_k]$ be a non-zero polynomial of total degree $d$. Then for $P = (P_1,\ldots,P_k)$ chosen uniformly at random in $\mathbb{F}_r^k$ the probability that $F(P_1,\ldots,P_k) = 0$ is at most $d/r$.*

**Proof:** If $k = 1$ then the result is standard. We prove the result by induction on $k$. Write

$$F(x_1,\ldots,x_k) = F_e(x_1,\ldots,x_{k-1})x_k^e + F_{e-1}(x_1,\ldots,x_{k-1})x_k^{e-1} + \cdots + F_0(x_1,\ldots,x_{k-1})$$

where $F_i(x_1,\ldots,x_{k-1}) \in \mathbb{F}_r[x_1,\ldots,x_{k-1}]$ has total degree $\leq d-i$ for $0 \leq i \leq e$ and $e \leq d$. If $P = (P_1,\ldots,P_{k-1}) \in \mathbb{F}_r^{k-1}$ is such that all $F_i(P) = 0$ then all $r$ choices for $P_k$ lead to a solution. The probability of this happening is at most $(d-e)/r$ (this is the probability that $F_e(P) = 0$). On the other hand, if some $F_i(P) \neq 0$ then there are at most $e$ choices for $P_k$ that give a root of the polynomial. The total probability is therefore $\leq (d-e)/r + e/r = d/r$. □

**Theorem 13.4.5.** *Let $G$ be a cyclic group of prime order $r$. Let $A$ be a generic algorithm for the DLP in $G$ that makes at most $m < \sqrt{r} - 2$ oracle queries. Then the probability, over uniformly chosen $a \in \mathbb{Z}/r\mathbb{Z}$ and uniformly chosen encoding function $\sigma : G \to \{0,1\}^{\lceil t\log(r)\rceil}$, that $A(\sigma(g),\sigma(g^a)) = a$ is at most $(m+2)^2/(2r)$.*

**Proof:** Instead of choosing a random encoding function in advance, the method of proof is to create the encodings "on the fly". The algorithm to produce the encodings is called the simulator. We also do not choose the instance of the DLP until the end of the game. The simulation will be perfect unless a certain bad event happens, and we will analyse the probability of this event.

Let $S = \{0,1\}^{\lceil t\log(r)\rceil}$. The simulator begins by uniformly choosing two distinct $\sigma_1, \sigma_2$ in $S$ and running $A(\sigma_1,\sigma_2)$. Algorithm $A$ treats $\sigma_1 = \sigma(g)$ and $\sigma_2 = \sigma(h)$ as an instance of the discrete logarithm problem for some $g, h \in G$ and some encoding function $\sigma$, but it is not necessary for the simulator to fix in advance the values for $g$ and $h$.

It is necessary to ensure that the encodings are consistent with the group operations. This cannot be done perfectly without choosing $g$ and $h$, but the following idea takes care of "trivial" consistency. From now on we use additive notation; one can think of

this either as writing $G$ as the additive group $(\mathbb{Z}/r\mathbb{Z}, +)$, or writing group elements as $g^{F(x)}$. The simulator maintains a list of pairs $(\sigma_i, F_i)$ where $\sigma_i \in S$ and $F_i \in \mathbb{F}_r[x]$. The initial values are $(\sigma_1, 1)$ and $(\sigma_2, x)$. Whenever $A$ makes an oracle query on $(\sigma_i, \sigma_j)$ the simulator computes $F = F_i - F_j$. If $F$ appears as $F_k$ in the list of pairs then the simulator replies with $\sigma_k$ and does not change the list. Otherwise, a value $\sigma \in S$ distinct from the previously used values is chosen uniformly at random, $(\sigma, F)$ is added to the simulator's list, and $\sigma$ is returned to $A$.

After making at most $m$ oracle queries $A$ outputs $b \in \mathbb{Z}/r\mathbb{Z}$. The simulator now chooses $a$ uniformly at random in $\mathbb{Z}/r\mathbb{Z}$. Algorithm $A$ wins if $b = a$.

Let the simulator's list contain precisely $k$ polynomials $\{F_1(x), \ldots, F_k(x)\}$ for some $k \leq m + 2$. Let $E$ be the event that $F_i(a) = F_j(a)$ for some pair $1 \leq i < j \leq k$. If event $E$ occurs then the simulation has not been consistent with the initial values $\sigma_1 = \sigma(g)$ and $\sigma_2 = \sigma(g^a)$. On the other hand, if event $E$ does not occur then the simulation has been consistent and valid.

The probability that $A$ wins is

$$\Pr(A \text{ wins } | E) \Pr(E) + \Pr(A \text{ wins } | \neg E) \Pr(\neg E). \tag{13.3}$$

For each pair $1 \leq i < j \leq k \leq m + 2$ the probability that $(F_i - F_j)(a) = 0$ is at most $1/r$ by Lemma 13.4.4. Hence, the probability of event $E$ is at most $k(k-1)/(2r)$. On the other hand, if event $E$ does not occur then $a$ must have been sampled from the set $\mathcal{X}$ of values for $a$ for which $F_i(a) \neq F_j(a)$ for all $1 \leq i < j \leq k$. Let $N = \#\mathcal{X}$ and note that $N \geq r - k(k-1)/2$ and $\Pr(\neg E) = 1 - \Pr(E) = N/r$. Since the simulation is perfect when event $E$ does not occur, we suppose that $A$ outputs a guess for the discrete logarithm that is consistent with the queries it made. This means $\Pr(A \text{ wins } | \neg E) \leq 1/N$.

Putting it all together, the probability that $A$ wins is at most

$$\Pr(E) + \Pr(A \text{ wins } | \neg E) \Pr(\neg E) \leq \frac{(m+1)(m+2)}{2r} + \frac{1}{N}\frac{N}{r} = \frac{m^2 + 3m + 4}{2r}.$$

This gives the result. □

**Exercise 13.4.6.** Prove Theorem 13.4.5 using Maurer's model for generic algorithms. [Hint: The basic method of proof is exactly the same. The difference is in formulation and analysis of the success probability.]

**Corollary 13.4.7.** *Let $A$ be a generic algorithm for the DLP. If $A$ succeeds with noticeable probability $1/\log(r)^c$ for some $c > 0$ then $A$ must make $\Omega(\sqrt{r/\log(r)^c})$ oracle queries.*

## 13.5 Generalised Discrete Logarithm Problems

A number of generalisations of the discrete logarithm problem have been proposed over the years. The motivation for such problems varies: sometimes the aim is to enable new cryptographic functionalities; other times the aim is to generate hard instances of the DLP more quickly than previous methods.

**Definition 13.5.1.** Let $G$ be a finitely generated Abelian group. The **multidimensional discrete logarithm problem** or **representation problem**[4] is: given $g_1, g_2, \ldots, g_l, h \in G$ and $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_l \subseteq \mathbb{Z}$ to find $a_j \in \mathcal{S}_j$ for $1 \leq j \leq l$, if they exist, such that

$$h = g_1^{a_1} g_2^{a_2} \cdots g_l^{a_l}.$$

[4] This computational problem seems to be first explicitly stated in the work of Brands [97] from 1993, in the case $\mathcal{S}_i = \mathbb{Z}$.

The **product discrete logarithm problem**[5] is: given $g, h \in G$ and $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_l \subseteq \mathbb{Z}$ to find $a_j \in \mathcal{S}_j$ for $1 \leq j \leq l$, if they exist, such that

$$h = g^{a_1 a_2 \cdots a_l}.$$

**Remark 13.5.2.** A natural variant of the product DLP is to compute only the product $a_1 a_2 \cdots a_l$ rather than the $l$-tuple $(a_1, \ldots, a_l)$. This is just the DLP with respect to a specific instance generator (see the discussion in Section 2.1.2). Precisely, consider an instance generator that, on input a security parameter $\kappa$, outputs a group element $g$ of prime order $r$ and then chooses $a_j \in \mathcal{S}_j$ for $1 \leq j \leq l$ and computes $h = g^{a_1 a_2 \cdots a_l}$. The stated variant of the product DLP is the DLP with respect to this instance generator.

Note that the representation problem can be defined whether or not $G = \langle g_1, \ldots, g_l \rangle$ is cyclic. The solution to Exercise 13.5.4 applies in all cases. However, there may be other ways to tackle the non-cyclic case (e.g., exploiting efficiently computable group homomorphisms, see [231] for example), so the main interest is the case when $G$ is cyclic of prime order $r$.

**Example 13.5.3.** The representation problem can arise when using the GLV method (see Section 11.3.3) with intentionally small coefficients. In this case, $g_2 = \psi(g_1)$, $\langle g_1, g_2 \rangle$ is a cyclic group of order $r$, and $h = g_1^{a_1} g_2^{a_2}$ where $0 \leq a_1, a_2 < w \leq \sqrt{r}$).

The number of possible choices for $h$ in both the representation problem and product DLP is at most $\prod_{j=1}^{l} \#\mathcal{S}_j$ (it could be smaller if the same $h$ can arise from many different combinations of $(a_1, \ldots, a_l)$). If $l$ is even and $\#\mathcal{S}_j = \#\mathcal{S}_1$ for all $j$ then there is an easy time/memory tradeoff algorithm requiring $O(\#\mathcal{S}_1^{l/2})$ group operations.

**Exercise 13.5.4.** Write down an efficient BSGS algorithm to solve the representation problem. What is the running time and storage requirement?

**Exercise 13.5.5.** Give an efficient BSGS algorithm to solve the product DLP. What is the running time and storage requirement?

It is natural to ask whether one can do better than the naive baby-step-giant-step algorithms for these problems, at least for certain values of $l$. The following result shows that the answer in general turns out to be "no".

**Lemma 13.5.6.** *Assume $l$ is even and $\#\mathcal{S}_j = \#\mathcal{S}_1$ for all $2 \leq j \leq l$. A generic algorithm for the representation problem with noticeable success probability $1/\log(\#\mathcal{S}_1)^c$ needs $\Omega(\#\mathcal{S}_1^{l/2}/\log(\#\mathcal{S}_1)^{c/2})$ group operations.*

**Proof:** Suppose $A$ is a generic algorithm for the representation problem. Let $G$ be a group of order $r$ and let $g, h \in G$. Set $m = \lceil r^{1/l} \rceil$, $\mathcal{S}_j = \{a \in \mathbb{Z} : 0 \leq a < m\}$ and let $g_j = g^{m^j}$ for $0 \leq j \leq l-1$. If $h = g^a$ for some $a \in \mathbb{Z}$ then the base $m$-expansion $a_0 + a_1 m + \cdots + a_{l-1} m^{l-1}$ is such that

$$h = g^a = \prod_{j=0}^{l-1} g_j^{a_j}.$$

Hence, if $A$ solves the representation problem then we have solved the DLP using a generic algorithm. Since we have shown that a generic algorithm for the DLP with success probability $1/\log(\#\mathcal{S}_1)^c$ needs $\Omega(\sqrt{r/\log(\#\mathcal{S}_1)^c})$ group operations, the result is proved. □

[5] The idea of using product exponents for improved efficiency appears in Knuth [343] where it is called the "factor method".

## 13.6 Low Hamming Weight DLP

Recall that the **Hamming weight** of an integer is the number of ones in its binary expansion.

**Definition 13.6.1.** Let $G$ be a group and let $g \in G$ have prime order $r$. The **low Hamming weight DLP** is: Given $h \in \langle g \rangle$ and integers $n, w$ to find a integer $a$ (if it exists) whose binary expansion has length $\leq n$ and Hamming weight $\leq w$ such that $h = g^a$.

This definition makes sense even for $n > \log_2(r)$. For example, squaring is faster than multiplication in most representations of algebraic groups, so it could be more efficient to compute $g^a$ by taking longer strings with fewer ones in their binary expansion.

Coppersmith developed a time/memory tradeoff algorithm to solve this problem. A thorough treatment of these ideas was given by Stinson in [591]. Without loss of generality we assume that $n$ and $w$ are even (just add one to them if not).

The idea of the algorithm is to reduce solving $h = g^a$ where $a$ has length $n$ and Hamming weight $w$ to solving $hg^{-a_2} = g^{a_1}$ where $a_1$ and $a_2$ have Hamming weight $w/2$. One does this by choosing a set $B \subset I = \{0, 1, \ldots, n-1\}$ of size $n/2$. The set $B$ is the set of possible bit positions for the bits of $a_1$ and $(I - B)$ is the possible bit positions for the bits of $a_2$. The detailed algorithm is given in Algorithm 15. Note that one can compactly represent subsets $Y \subseteq I$ as $n$-bit strings.

**Algorithm 15** Coppersmith's baby-step-giant-step algorithm for the low Hamming weight DLP

INPUT: $g, h \in G$ of order $r$, $n$ and $w$
OUTPUT: $a$ of bit-length $n$ and Hamming weight $w$ such that $h = g^a$, or $\perp$

```
Choose B ⊂ {0,...,n-1} such that #B = n/2
Initialise an easily searched structure (such as a binary tree, a heap, or a hash table) L
for Y ⊆ B : #Y = w/2 do
    Compute b = Σ_{j∈Y} 2^j and x = g^b
    store (x, Y) in L ordered according to first coordinate
end for
for Y ⊆ (I − B) : #Y = w/2 do
    Compute b = Σ_{j∈Y} 2^j and y = hg^{-b}
    if y = x for some (x, Y_1) ∈ L then
        a = Σ_{j∈Y∪Y_1} 2^j
        return a
    end if
end for
return ⊥
```

**Exercise 13.6.2.** Write down an algorithm, to enumerate all $Y \subset B$ such that $\#Y = w/2$, which requires $O(\binom{n/2}{w/2} n)$ bit operations.

**Lemma 13.6.3.** *The running time of Algorithm 15 is $O(\binom{n/2}{w/2})$ group operations and the algorithm requires $O(\binom{n/2}{w/2})$ group elements of storage.*

**Exercise 13.6.4.** Prove Lemma 13.6.3.

Algorithm 15 is not guaranteed to succeed, since the set $B$ might not exactly correspond to a splitting of the bit positions of the integer $a$ into two sets of Hamming weight $\leq w/2$. We now give a collection of subsets of $I$ that is guaranteed to contain a suitable $B$.

**Definition 13.6.5.** Fix even integers $n$ and $w$. Let $I = \{0, \ldots, n-1\}$. A **splitting system** is a set $\mathcal{B}$ of subsets of $I$ of size $n/2$ such that for every $Y \subset I$ such that $\#Y = w$ there is a set $B \in \mathcal{B}$ such that $\#(B \cap Y) = w/2$.

**Lemma 13.6.6.** *For any even integers $n$ and $w$ there exists a splitting system $\mathcal{B}$ of size $n/2$.*

**Proof:** For $0 \leq i \leq n-1$ define

$$B_i = \{i + j \pmod{n} : 0 \leq j \leq n/2 - 1\}$$

and let $\mathcal{B} = \{B_i : 0 \leq i \leq n/2 - 1\}$.

To show $\mathcal{B}$ is a splitting system, fix any $Y \subset I$ of size $w$. Define $\nu(i) = \#(Y \cap B_i) - \#(Y \cap (I - B_i)) \in \mathbb{Z}$ for $0 \leq i \leq n/2 - 1$. One can check that $\nu(i)$ is even, that $\nu(n/2) = -\nu(0)$ and that $\nu(i+1) - \nu(i) \in \{-2, 0, 2\}$. Hence, either $\nu(0) = 0$, or else the values $\nu(i)$ change sign at least once as $i$ goes from 0 to $n/2$. It follows that there exists some $0 \leq i \leq n/2$ such that $\nu(i) = 0$, in which case $\#(Y \cap B_i) = w/2$. □

One can run Algorithm 15 for all $n/2$ sets $B$ in the splitting system $\mathcal{B}$ of Lemma 13.6.6. This gives a deterministic algorithm with running time $O(n\binom{n/2}{w/2})$ group operations. Stinson proposes different splitting systems giving a deterministic algorithm requiring $O(w^{3/2}\binom{n/2}{w/2})$ group operations. A more efficient randomised algorithm (originally proposed by Coppersmith) is to randomly choose sets $B$ from the $\binom{n}{n/2}$ possible subsets of $\{0, \ldots, n-1\}$ of size $n/2$. Theorem 13.6.9 determines the expected running time in this case.

**Lemma 13.6.7.** *Fix a set $Y \subset \{0, \ldots, n-1\}$ such that $\#Y = w$. The probability that a randomly chosen $B \subseteq \{0, \ldots, n-1\}$ having $\#B = n/2$ satisfies $\#(Y \cap B) = w/2$ is*

$$p_{Y,B} = \binom{w}{w/2}\binom{(n-w)}{(n-w)/2} / \binom{n}{n/2}.$$

**Exercise 13.6.8.** Prove Lemma 13.6.7.

**Theorem 13.6.9.** *The expected running time for the low Hamming weight DLP when running Algorithm 15 on randomly chosen sets $B$ is $O(\sqrt{w}\binom{n/2}{w/2})$ exponentiations. The storage is $O(\binom{n/2}{w/2})$ group elements.*

**Proof:** We expect to repeat the algorithm $1/p_{Y,B}$ times. One can show, using the fact $2^k/\sqrt{2k} \leq \binom{k}{k/2} \leq 2^k\sqrt{2/\pi k}$, that $1/p_{Y,B} \leq c\sqrt{w}$ for some constant (see Stinson [591]). The result follows. □

**Exercise 13.6.10.** As with all baby-step-giant-step methods, the bottleneck for this method is the storage requirement. Show how to modify the algorithm for the case where only $M$ group elements of storage are available.

**Exercise 13.6.11.** Adapt Coppersmith's algorithm to the DLP for low weight signed expansions (for example, NAFs, see Section 11.1.1).

All the algorithms in this section have large storage requirements. An approach due to van Oorschot and Wiener for solving such problems using less storage is presented in Section 14.8.1.

## 13.7 Low Hamming Weight Product Exponents

Let $G$ be an algebraic group (or algebraic group quotient) over $\mathbb{F}_p$ ($p$ small) and let $g \in G(\mathbb{F}_{p^n})$ with $n > 1$. Let $\pi_p$ be the $p$-power Frobenius on $G$, acting on $G$ as $g \mapsto g^p$. Hoffstein and Silverman [290] proposed computing random powers of $g$ efficiently by taking products of low Hamming weight Frobenius expansions.

In particular, for Koblitz elliptic curves (i.e., $p = 2$) they suggested using three sets and taking $\mathcal{S}_j$ for $1 \leq j \leq 3$ to be the set of Frobenius expansions of length $n$ and weight 7. The baby-step-giant-step algorithm in Section 13.5 applies to this problem, but the running time is not necessarily optimal since $\#\mathcal{S}_1 \#\mathcal{S}_2 \neq \#\mathcal{S}_3$. Kim and Cheon [338] generalised the results of Section 13.6 to allow a more balanced time/memory tradeoff. This gives a small improvement to the running time.

Cheon and Kim [134] give a further improvement to the attack, which is similar to the use of equivalence classes in Pollard rho (see Section 14.4). They noted that the sets $\mathcal{S}_j$ in the Hoffstein-Silverman proposal have the property that for every $a \in \mathcal{S}_j$ there is some $a' \in \mathcal{S}_j$ such that $g^{a'} = \pi_p(g^a)$. In other words, $\pi_p$ permutes $\mathcal{S}_j$ and each element $a \in \mathcal{S}_j$ lies in an orbit of size $n$ under this permutation. Cheon and Kim define a unique representative of each orbit of $\pi_p$ in $\mathcal{S}_j$ and show how to speed up the BSGS algorithm in this case by a factor of $n$.

**Exercise 13.7.1.** ★ Give the details of the Cheon-Kim algorithm. How many group operations does the algorithm perform when $n = 163$ and three sets with $w = 7$ are used?

## 13.8 Wagner's Generalised Birthday Algorithm

This section presents an algorithm due to Wagner [625] (though a special case was discovered earlier by Camion and Patarin), which has a similar form to the baby-step-giant-step algorithm. This algorithm is not useful for solving the DLP in groups of relevance to public key cryptography, but it is an example of how a non-generic algorithm can beat the birthday bound. Further examples of non-generic algorithms that beat the birthday bound are given in Chapter 15. For reasons of space we do not present all the details.

**Definition 13.8.1.** Suppose one is given large sets $L_j$ of $n$-bit strings, for $1 \leq j \leq l$. The **$l$-sum problem** is to find $x_j \in L_j$ for $1 \leq j \leq l$ such that

$$x_1 \oplus x_2 \oplus \cdots \oplus x_l = 0, \tag{13.4}$$

where 0 denotes the $n$-bit all zero string.

The $l$-sum problem is easy if $0 \in L_i$ for all $0 \leq i \leq l$. Another relatively easy case is if $l$ is even and $L_{2i-1} \cap L_{2i} \neq \varnothing$ for all $1 \leq i \leq l/2$. Hence, the $l$-sum problem is of most interest when the sets $L_i$ are chosen independently and at random. By the coupon collector theorem (Example A.14.3) one expects a solution to exist when $\#L_1 \cdots \#L_l > 2^n \log(n)$ if the $L_j$ are sufficiently random.

**Exercise 13.8.2.** Give a baby-step-giant-step algorithm to solve this problem when $l = 2$.

**Exercise 13.8.3.** Give an example of sets $L_1, L_2$ of $n$-bit strings such that $\#L_1, \#L_2 > 2^{\lceil n/2 \rceil}$ but there is no solution to the 2-sum problem.

We now sketch the method in the case $l = 4$. Let $m = \lceil n/3 \rceil$. It will be necessary to assume that $\#L_j \#L_{j+1} \geq 2^{2m}$ (e.g., $\#L_j \geq 2^m$) for each $j = 1, 3$, so this method is

not expected to work if $\#L_j \approx 2^{n/4}$ for all $1 \leq j \leq 4$. Define $\mathrm{LSB}_m(x) =$ the $m$-least significant bits of the bit-string $x$.

The first step is to form the sets

$$L_{j,j+1} = \{(x_j, x_{j+1}) \in L_j \times L_{j+1} : \mathrm{LSB}_m(x_j \oplus x_{j+1}) = 0\}$$

for $j = 1, 3$. These sets can be formed efficiently. For example, to build $L_{1,2}$: sort the list $L_1$ (at least, sort with respect to the $m$ least significant bits of each string), then for each $x_2 \in L_2$ test whether there exists $x_1 \in L_2$ such that $\mathrm{LSB}_m(x_1) = \mathrm{LSB}_m(x_2)$. If the sets $L_i$ are sufficiently random then it is reasonable to suppose that the size of $L_{j,j+1}$ is $\#L_j \#L_{j+1}/2^m \geq 2^m$. To each pair $(x_j, x_{j+1}) \in L_{j,j+1}$ we can associate the $(n-m)$-bit string obtained by removing the $m$ least significant bits of $x_j \oplus x_{j+1}$.

The second step is to find $(x_1, x_2) \in L_{1,2}$ and $(x_3, x_4) \in L_{3,4}$ such that $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$. This is done by sorting the $(n-m)$-bit truncated $x_1 \oplus x_2$ corresponding to $(x_1, x_2) \in L_{1,2}$ and then, for each $(x_3, x_4) \in L_{3,4}$ testing whether the $(n-m)$-bit truncated $x_3 \oplus x_4$ is in the list. Since $\#L_{1,2}, \#L_{3,4} \geq 2^m$ and $n - m \approx 2m$ then, if the sets $L_{j,j+1}$ are sufficiently random, there is a good chance that a solution will exist.

The above arguments lead to the following heuristic result.

**Heuristic 13.8.4.** Let $n \in \mathbb{N}$ and $m = \lceil n/3 \rceil$. Suppose the sets $L_i \subset \{0,1\}^n$ for $1 \leq i \leq 4$ are randomly chosen and that $\#L_j \#L_{j+1} \geq 2^{2m}$ for $j = 1, 3$. Then Wagner's algorithm should find a solution $(x_1, \ldots, x_4)$ to equation (13.4) in the case $l = 4$. The running time is $\tilde{O}(2^m) = \tilde{O}(2^{n/3})$ bit operations and the algorithm requires $\tilde{O}(2^m) = \tilde{O}(2^{n/3})$ bits of storage.

The algorithm has "cube root" complexity, which beats the usual square-root complexity bound for such problems. The reason is that we are working in the group $(\mathbb{F}_2^n, +)$ and the algorithm is *not* a generic algorithm: it exploits the fact that the group operation and group representation satisfy the property $\mathrm{LSB}_m(x) = \mathrm{LSB}_m(y) \Leftrightarrow \mathrm{LSB}_m(x \oplus y) = 0$.

The algorithm is not expected to succeed in the case when $\#L_j \approx 2^{n/4}$ since it is finding a solution to equation (13.4) of a very special form (namely, that $\mathrm{LSB}_m(x_1 \oplus x_2) = \mathrm{LSB}_m(x_3 \oplus x_4) = 0$).

**Exercise 13.8.5.** Generalise this algorithm to the case $l = 2^k$. Show that the algorithm is heuristically expected to require time and space $\tilde{O}(l 2^{n/(1+k)})$. What is the minimum size for the $L_j$ (assuming they are all of equal size)?

**Exercise 13.8.6.** Wagner's algorithm is deterministic, but it is not guaranteed to succeed on a given input. How can one "randomise" Wagner's algorithm so that any instance (with large enough lists) can be solved efficiently with high probability?

The 4-sum problem can be put into a more general framework: Let $\mathcal{S}, \mathcal{S}'$ and $\mathcal{S}''$ be sets such that $\#\mathcal{S}' = N$, fix an element $0 \in \mathcal{S}''$, let $f_1, f_2 : \mathcal{S} \times \mathcal{S} \to \mathcal{S}'$ and $f : \mathcal{S}' \times \mathcal{S}' \to \mathcal{S}''$ be functions. Let $L_1, L_2, L_3, L_4 \subset \mathcal{S}$ be randomly chosen subsets of size $\#L_i \approx N^{1/3}$ and suppose one wants to find $x_j \in L_j$ for $1 \leq j \leq 4$ such that

$$f(f_1(x_1, x_2), f_2(x_3, x_4)) = 0.$$

Wagner's algorithm can be applied to solve this problem if there is a distinguished set $\mathcal{D} \subset \mathcal{S}'$ such that the following five conditions hold:

1. $\#\mathcal{D} \approx N^{2/3}$.
2. $\Pr(f(y_1, y_2) = 0 : y_1, y_2 \leftarrow \mathcal{D}) \approx N^{-2/3}$.

3. $\Pr(f_1(x_1,x_2)\in\mathcal{D} : x_1 \leftarrow L_1, x_2 \leftarrow L_2) \approx \Pr(f_2(x_3,x_4)\in\mathcal{D} : x_3 \leftarrow L_3, x_4 \leftarrow L_4) \approx N^{-1/3}$.

4. For $j = 1, 2$ one can determine, in $\tilde{O}(N^{1/3})$ bit operations, lists

$$L_{J,J+1} = \{(x_J, x_{J+1}) \in L_J \times L_{J+1} : f_j(x_J, x_{J+1}) \in \mathcal{D}\}$$

   where $J = 2j - 1$.

5. Given $L_{1,2}$ and $L_{3,4}$ as above one can determine, in $\tilde{O}(N^{1/3})$ bit operations,

$$\{((x_1,x_2),(x_3,x_4)) \in L_{1,2} \times L_{3,4} : f(f_1(x_1,x_2), f_2(x_3,x_4)) = 0\}.$$

**Exercise 13.8.7.** Show that the original Wagner algorithm for $\mathcal{S} = \mathcal{S}' = \mathcal{S}'' = \{0,1\}^n$ fits this formulation. What is the set $\mathcal{D}$?

**Exercise 13.8.8.** Describe Wagner's algorithm in the more general formulation.

**Exercise 13.8.9.** Let $\mathcal{S} = \mathcal{S}' = \mathcal{S}''$ be the additive group $(\mathbb{Z}/N\mathbb{Z}, +)$ of integers modulo $N$. Let $L_1, L_2, L_3, L_4 \subset \mathbb{Z}/N\mathbb{Z}$ be such that $\#L_i \approx N^{1/3}$. Let $f_1(x_1,x_2) = f_2(x_1,x_2) = f(x_1,x_2) = x_1 + x_2 \pmod N$. Let $\mathcal{D} = \{y \in \mathbb{Z} : -N^{2/3}/2 \leq y \leq N^{2/3}/2\}$. Show that the above 5 properties hold in this setting. Can you think of any better method to solve the problem in this setting?

**Exercise 13.8.10.** Let $\mathcal{S} \subseteq \mathbb{Z}$ and $\mathcal{S}' = \mathcal{S}'' = \mathbb{F}_p$. Let $(g_1, g_2, g_3, g_4, h)$ be an instance of the representation problem in $\mathbb{F}_p^*$. Consider the functions

$$f_1(x_1,x_2) = g_1^{x_1} g_2^{x_2} \pmod p, \quad f_2(x_3,x_4) = h g_3^{-x_3} g_4^{-x_4} \pmod p$$

and $f(y_1,y_2) = y_1 - y_2 \pmod p$. Finding a solution to $f(f_1(x_1,x_2), f_2(x_3,x_4)) = 0$ solves the representation problem.

Let $m = \log_2(p)/3$ and define $\mathrm{LSB}_m(y)$ for $y \in \mathbb{F}_p$ by representing $y$ as an integer in the range $0 \leq y < p$ and outputting the $m$ least significant bits. Let $\mathcal{D} = \{y \in \mathbb{F}_p^* : \mathrm{LSB}_m(y) = 0\}$. Explain that the property 4 of the above list does not seem to hold for this example.